Rein Ruutiainen

Työkuorman suunnittelu- ja hallintatyökalu

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tietotekniikka
Insinöörityö
12.3.2012

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Rein Ruutiainen Työkuorman suunnittelu- ja hallintatyökalu 41 sivua 12.3.2012
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaajat	toimitusjohtaja Terho Norja lehtori Olli Hämäläinen

Tämä insinöörityö oli osa IPSS Oy:n panosta vuoden 2011 Cloud Software -tutkimusohjelmaan, jota järjesti Tieto- ja viestintäteollisuuden tutkimus TIVIT Oy. Cloud Software -ohjelma käsittää useamman tutkimuskohteen ja tämä insinöörityö oli tarkemmin mukana Lean Software Enterprise -osiossa, jossa keskityttiin tutkimaan ketterien menetelmien hyödyntämistä ja kehittämistä.

Insinöörityön tavoitteena oli tehdä työkalu, joka tukisi työkuorman suunnittelua ja hallintaa päivä- ja viikkotasolla. Haluttiin mahdollisuus suunnitella minä päivänä tekee mitäkin työtehtävää sekä kuinka kauan ja samalla näytettäisiin myös mitä tapaamisia, palavereja ja työpajoja on milloinkin. Tavoitteena loppujen lopuksi oli auttaa hahmottamaan, milloin olisi aikaa tehdä mikäkin tehtävä kaikkien muiden työasiatapahtumien ohella.

Työkaluun tehtiin oma kalenteri, jossa näkyvät kaikki eri tapaamiset, palaverit ja työpajat, jotka noudetaan IPSS iSteer Contact CRM -ohjelmiston kalenterista sekä voi suunnitella työtehtäviään työkalun kalenterin päiville, joiden tiedot noudetaan suomalaisesta avoimen lähdekoodin Agilefant-projektinhallintaohjelmistosta. Insinöörityössä toteutettu työkalu lukee näiden tietolähteitä ja tämä integraatio mahdollistaa oman henkilökohtaisen työkuormakalenterin luonnin. Työkalun työkuormakalenteri näyttää, kuinka paljon työkuormaa on yhteensä kerääntynyt päiville ja viikoille. Työkalusta tehtiin selainpohjainen sovellus, jossa toteutettiin palvelinpuoli Java Spring -ohjelmistokehyksellä ja tietokantayhteydet Hibernate-ohjelmistokehyksellä. Käyttöliittymä tehtiin jQueryä käyttäen. Käyttöliittymän suunnittelu kuului myös insinöörityöhön.

Projektissa saavutettiin asetetut vaatimukset ensimmäisen version toteutukselle. Työkalu on käytettävissä yrityksen sisällä, ja käyttökokemuksia kuunnellaan potentiaalista jatkokehitystä ajatellen. Tämänhetkinen näkemys seuraavasta versiosta olisi muitten Agilefantin toimintojen integroiminen, kuten tuntikirjauksien tekeminen ja seuraaminen suoraan työkalusta.

Avainsanat	työkuorman hallinta, Agilefant-integraatio, verkko-ohjelmointi, käyttöliittymäsuunnittelu, Java Spring, Hibernate, jQuery

Metropolia

Abstract

Author Title Number of Pages Date	Rein Ruutiainen Workload planning and management tool 41 pages 12 March 2012
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructors	Terho Norja, CEO Olli Hämäläinen, Senior Lecturer

This thesis was part of IPSS Oy's contribution to TIVIT Oy's Cloud Software research program in 2011. In general, the Cloud Software program includes several research programs and this thesis was part of the Lean Software Enterprise program which concentrates on further developing and utilizing agile software development approaches.

The goal was to make a tool to support workload planning and management on daily and weekly levels so that the user can allocate work assignments to any working day and also have meetings and workshop calendar markings shown in the same interface. The tool has its own calendar where the planning is meant to take place. There the user can see the total workload for a particular day or week. Ultimately this lets the user to know when there is time to do a particular task.

The tool has its own calendar showing one's own customer meetings and workshops which are read from the IPSS iSteer Contact CRM software's calendar. Secondly, work assignments can be planned in the tool's calendar and one's own current assignments are fetched from a Finnish open source project management program called Agilefant. The tool integrates data from these both and makes it possible to plan one's own workload calendar, and this shows the total workload one has for a day and a whole week.

The project was programmed with the Spring framework for server-side functionalities and the Hibernate framework for database connections. Most of the project was carried out with client-side programming in jQuery. Also designing the interface was part of the project.

All the features that were planned for the first version of the tool were achieved. The tool is now available on the company intranet and user experiences are gathered for potential follow-up development in the future. Current plans for the future would consist of integrating more Agilefant features such as saving and reading Agilefant hour loggings.

Keywords	workload management, Agilefant integration, web programming, GUI design, Java Spring, Hibernate, jQuery

Sisällys

Lyhenteet

1 Johdanto 1

2 Suunnittelu 3

3 Projektinhallinta Agilefantilla 5

3.1 Työkuorman hallinta 5
3.2 Puutteet työkuorman hallinnassa 7

4 Käytetty tekniikka 8

4.1 Spring MVC 8
4.2 Käyttöliittymän tekniikka 10

5 Tietolähteiden integraatio 12

5.1 Käytetyt suunnittelumallit 12
5.2 Agilefant-integraatio 13
5.3 Työsuunnitelman tallennus 19
5.4 IPSS iSteer Contact -integraatio 20

6 Toteutus 23

6.1 Ulkoasu 23
6.2 Käyttäjätietojen hakeminen 24
6.3 Puunäkymän muodostaminen 26
6.4 Työkuormakalenterin viikkonäkymä 28
6.5 Tehtävän sijoittaminen puunäkymästä 30
6.6 Sijoitusdialogi 32
6.7 Sijoitusdialogin virheiden käsittely 35
6.8 Tehtävän uudelleen siirto ja muokkaus 36

7 Yhteenveto 39

Lähteet 40

Lyhenteet ja käsitteet

Agilefant	Aalto-yliopiston tekniikan laitoksen ohjelmistoliiketoiminnan ja -tuotannon laboratorion eli SoberIT:n kehittämä projektinhallintatyökalu ketterille menetelmille. [3.]
AJAX	Asynchronous JavaScript And XML on verkkosovellus tekniikka, jolla voi keskustella palvelimen kanssa ilman, että tarvitsee ladata sivua uudestaan.
Annotaatio	Liittää metadataa kohteeseen.
Asiakkuudenhallinta	Keskitytään asiakaslähtöiseen ajattelutapaan organisaatiossa sekä siihen liittyvissä tietojärjestelmissä.
Backlog	Tuotteen työluettelo, joka sisältää kaikki tuotteelle määritetyt vaatimukset.
BO	Business Object sisältää logiikkaa, joka koskee tietokantoja. [13.]
CRM	Customer relationship management eli asiakkuudenhallinta.
CSS	Cascading Style Sheets on tekniikka hallita verkkosivustojen ulkoasua antamalla tyyliohjeita.
DAO	Data Access Object on suunnittelumalli, jossa ohjataan tietokantayhteyksiä. Tietojen tallennus, poistaminen ja lataaminen toteutetaan täällä. [14.]
Dependency Injection	Suunnittelumalli, jossa on ajatuksena tuoda luokan olio attribuuttien instanssit ulkoapäin, kuten rakentajan välityksellä. [7; 20, s. 231–234.]
DTO	Data Transfer Object on malli-tyyppinen luokka, joka edustaa tietokantatauluja attribuuteillaan. Ei sisällä logiikkaa. [12.]
Hibernate ORM	Javalle tehty tietokantojen olio-relaatio -kuvauksen ohjelmistokehys. Tekee mahdolliseksi muuttaa tietokantatauluja Java olioiksi ja toisinpäin. [6; 20, s. 20–21]
HTML	Hypertext Markup Language on yleisesti käytetty verkkosivujen kuvauskieli.
iSteer Contact CRM	IPSS Oy:n kehittämä Salesforce.com-pohjainen asiakkuudenhallinta ohjelmisto.

Iteraatio	Ketterissä menetelmissä käytetty yhdestä neljään viikkoon kestävä ohjelmistokehityksen ajanjakso, jossa on tarkoitus saavuttaa valmiiksi iteraatiolle asetetut ohjelman toiminnallisuudet. [18.]
Java	Virtuaalikoneella toimiva laitteistoriippumaton oliopohjainen ohjelmointikieli.
JavaBean	Java-luokka, josta löytyy parametriton rakentaja, kaikille attribuuteille get- ja set-metodit sekä toteuttaa java.io.Serializable. [16.]
JavaScript	Dynaamisesti tyypitetty ja tulkattava oliopohjainen komentosarjakieli.
jQuery	JavaScript-kirjasto, joka auttaa tekemään JavaScriptin käytöstä tehokkaampaa. [5.]
JSON	JavaScript Object Notation on tiedonsiirtomuoto.
Ketterät menetelmät	Ohjelmistotuotantoprojekteissa käytettyjä menetelmiä, joissa keskitytään ensisijaisesti toiminnallisen ohjelman tuottamiseen ennemmin kuin kokonaisvaltaiseen dokumentointiin. [17.]
Käsittelijä	Käsittelijä reagoi ja vastaa käyttäjän toimintoihin ja muokkaa malleja. Katso MVC-arkkitehtuuri.
LoTo	Insinöörityön aihe ja työkuorman suunnittelu- ja hallintatyökalu. Nimi tulee englanninkielisen nimensä Load planning and management tool lyhennyksestä Load tool.
Malli	Malli toimii tietokantakuvauksena ja käytetään tallentamiseen. Katso MVC-arkkitehtuuri.
MVC-arkkitehtuuri	Model, view, controller eli malli, näkymä, käsittelijä. MVC on ohjelmistoarkkitehtuurityyli, jossa ohjelmakoodi jaetaan kolmeen toisista eroavaan osaan.
MySQL	Relaatiotietokannan hallintaohjelmisto.
Näkymä	Edustaa käyttöliittymän ulkoasua. Katso MVC-arkkitehtuuri.
Ohjelmistokehys	Työkalupaketti tai runko rakennettavalle ohjelmalle, jossa on valmiiksi toteutettu tarpeelliset tekniikat, jotka muutoin tulisi itse ohjelmoida.

ORM	Object-relation mapping. Katso Hibernate ORM.
POJO	Plain Old Java Object edustaa Java-oliota, joka on puhdasta Javaa ilman riippuvuuksia suuriin ulkoisiin kirjastoihin tai ohjelmistokehyksiin. [16.]
Spring bean	Springin hallintaan asetettu olio. Nimi viittaa JavaBeaneihin, mutta Spring bean voi olla millainen olio vain, myös JavaBean. Voidaan asettaa Springin XML-tiedostoissa tai annotaatioilla. Hyödynnetään muun muassa sijoittamalla *Dependency Injection* -tekniikkaa käyttäen muuttujien arvoiksi. [7; 16.]
Spring Framework	Java-ohjelmistokehys, joka sisältää useita ohjelmointia tehostavia piirteitä ja lyhentää tarvittavaa pohjakoodia. [4.]
Tarina	Ketterissä menetelmissä käytetty termi kuvaamaan toiminnallisuutta. Agilefantissa tarinan alle voi laittaa tehtäviä. [19.]
Tehtävä	Agilefantissa tämä on *task*, joka vastaa yksittäistä työtehtävää.

1 Johdanto

IPSS - Intelligent Precision Solutions and Services Oy, jolle insinöörityö tehtiin, perustettiin vuonna 1999, ja se keskittyy asiakasjohtamisen ja -hallinnan teknologiaratkaisuihin ja liiketoiminnan kehittämiseen asiakastietoja hyödyntämällä, kuten kokoamalla asiakkaalle asiakastyöskentely-ympäristön, joka sisältää asiakastiedot, työkalut ja toimintamallit.

Tämä insinöörityö oli osa IPSS Oy:n panosta vuoden 2011 Cloud Software -tutkimusohjelmaan, jota järjesti Tieto- ja viestintäteollisuuden tutkimus TIVIT Oy. Cloud Software -ohjelma käsittää useamman tutkimuskohteen ja tämä insinöörityö oli tarkemmin mukana Lean Software Enterprise -osiossa, jossa keskityttiin tutkimaan ketterien menetelmien hyödyntämistä ja kehittämistä. [1; 2]

Ohjelmistokehitysalalla yhdelle henkilölle saattaa kertyä paljon töitä tehtäväksi lyhyellä ajalla, ja moni tehtävä voi vielä olla eri projektista. Tällaisia tilanteita varten on hyvä suunnitella etukäteen, mitä aikoo tehdä meneillään olevan viikon sekä muutaman tulevan viikon aikana ja samalla pitää huolta, että suosii kiireellisempiä työtehtäviä, ettei minkään tehtävän määräaika mene yllättäen umpeen. Joillain on myös työnsä ohella paljon palavereja, asiakastapaamisia ja työpajoja. Nämä tapahtumat tulee tietenkin myös ottaa huomioon, kun suunnittelee työkalenteriaan, jotta tietäisi, milloin voi ehtiä tekemään kiireellisimmät työnsä tapaamisiensa ulkopuolella. Hyvin suunniteltu työkalenteri huomioi, ettei ota viikolle enemmän työkuormaa kuin viikolla ehtii tehdä ottaen huomioon, että sovitut tapaamiset ja kokoontumiset voivat viedä huomattavasti aikaa. Samalla tulee katsoa, mitkä ovat kiireisimpiä töitä. Tämän tuloksena voisi saada työtehoa parannettua.

Insinöörityöni oli tehdä pilottiprojektina IPSS Oy:n sisäiseen käyttöön työkalu, jolla jokainen voisi suunnitella omaa työaikatauluaan paremmin sekä hallita ja seurata tehokkaammin työkuormaansa. Tällä hetkellä projekteja hallitaan Agilefant-projektinhallintatyökalulla ja tapahtumien sekä tapaamisien ajankohtia seurataan IPSS Oy:n omasta iSteer Contact CRM:n sisältämästä kalenterista. Ohjelma olisi näiden kahden tiedot integroiva työkuorman suunnittelu- ja hallintatyökalu. Ohjelman kehitysnimi

on LoTo, joka tulee englanninkielisen koko nimensä *Workload planning and management tool* lyhennyksestä *Load tool*.

LoTo:n käyttöliittymä koostuu työkuormakalenterista, jossa näkyy käyttäjän Contactin tapahtumat sekä omat Agilefantin tehtävät, joista Agilefantin tehtävien sijoittamisen voi itse suunnitella. Contactin tapahtumat sen sijaan ilmestyvät oikeille päiville suoraan eikä niitä LoTo:n kautta muokata. Agilefantin tehtävät näkyvät puunäkymässä, jossa sijaitsevat tuotteiden projektien iteraatioissa ja tarinoissa. Tehtäviä voi tästä siirtää hiirellä haluamilleen kalenterin päiville. Contactin tapahtumat ja Agilefantin tehtävät muodostavat yhdessä päiväkohtaisen työmäärän.

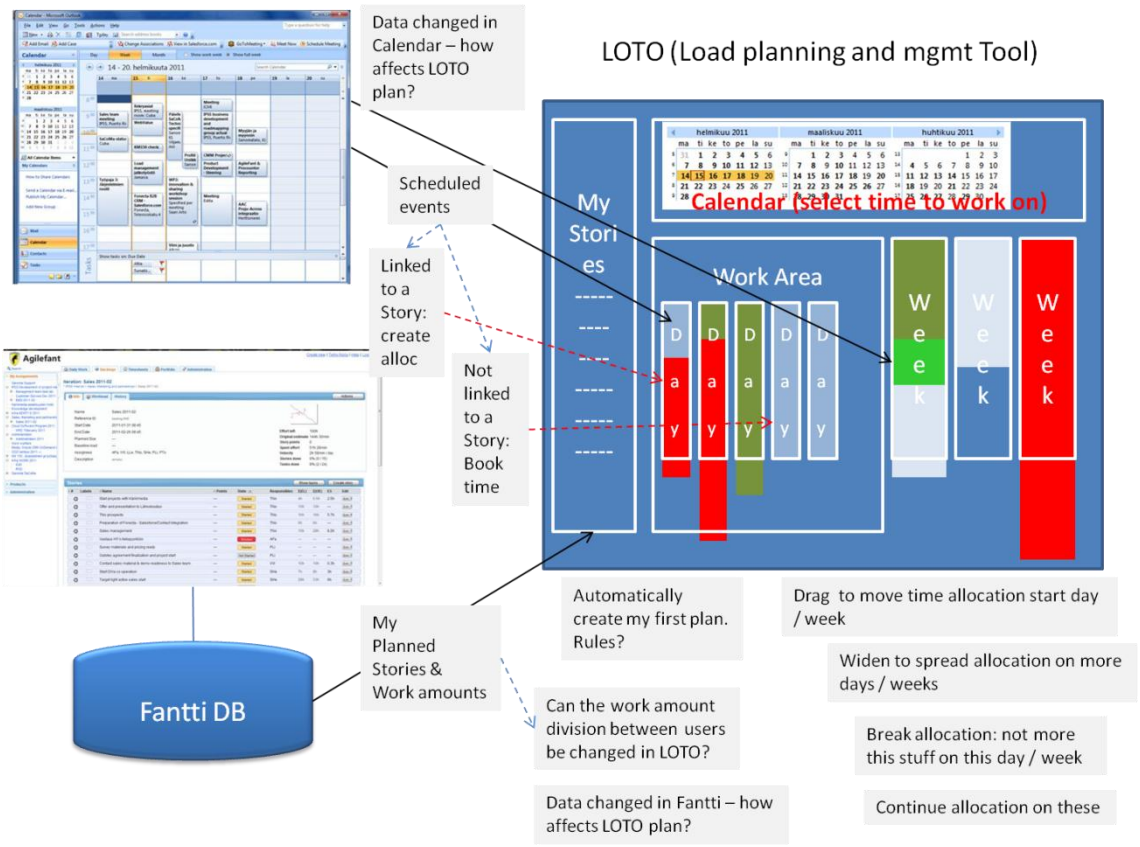
LoTo:n kalenterista ei katsota, mihin aikaan päivästä tehtävät tehdään tai tapaamiset pidetään, vaan sillä seurataan ja merkitään, mitä tehtäviä minäkin päivänä aikoo tehdä ja kuinka kauan. Annetaan oma arvio ajan käytöstä, eli kauanko aikoo tehtävää tehdä, ja näiden tuntiarvioiden mukaan piirretään työtehtävistä erikorkuisia pylväitä. Pylväät ladotaan päivän kohdalla päällekkäin yhdessä Contactin tapahtumien kanssa. Contactin tapahtumat ilmaistaan myös samalla tavalla työmäärän mukaisesti pylväinä. Nämä kaikki pylväät yhdessä korkeudellaan osoittavat, kuinka paljon töitä päivälle on jo varannut. Samoin myös koko viikosta on ilmaistu erikseen, paljonko viikolle on työkuormaa yhteensä suunniteltu palavereineen.

Teknisesti toteutin ohjelman verkko-ohjelmana, jossa palvelinpuoli on toteutettu Java Spring -ohjelmistokehyksellä ja käyttäjäpuoli on tehty JavaScriptin jQuery-kirjastolla [4; 5]. Tietokantayhteydet on hoidettu Hibernate-ohjelmistokehyksellä [6].

2 Suunnittelu

Mahdollisuuteni vaikuttaa projektin suunnitteluun ja etenemiseen oli hyvä. Kahden viikon välein pidettiin kolmen tai neljän hengen palaveri, jossa seurattiin projektin etenemistä ja suunniteltiin tulevaa.

Ennen kuin insinöörityöprojekti alkoi, oli sovellukselle alustavasti suunniteltu teknistä määritelmää ja ulkoasua. Tämä on esitetty kuviossa 1.



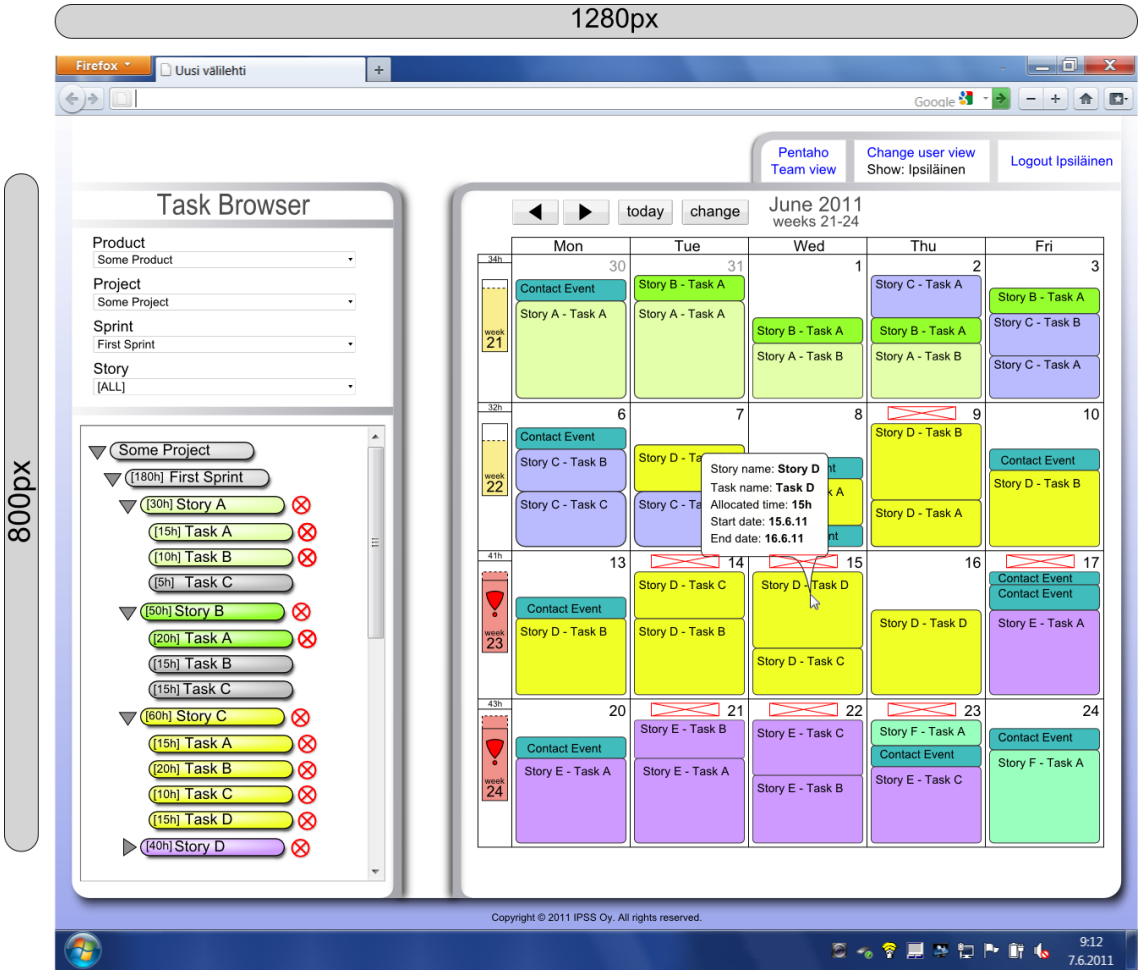
Kuvio 1. IPSS Oy:n alkuperäinen hahmotelma LoTo:sta, jonka pohjalta kehitys alkoi.

Haluttiin, että sijoittamalla Agilefantin tarinoita eri päiville ja viikoille voisi suunnitella omat työviikkonsa, sillä tätä ominaisuutta ei Agilefantissa itsessään tarjottu. Tämän lisäksi kaivattiin, että IPSS iSteer Contact CRM:ssä käytetyn kalenterin tapahtumat luettaisiin mukaan päivien ja viikkojen työkuormaan.

Kaikki työtehtävä sijoitukset kalenteriin haluttiin sijoitettavan hiirellä vetäen haluttuihin päiviin. Contactin kalenterista ladatut tapahtumat sijoittuisivat itsestään oikeille päiville

eikä näitä voisi siirtää pois tai muokata vaan tapahtuman siirtäminen tai muokkaaminen tulisi tehdä Contactissa itse ja tämä heijastuisi itsestään LoTo:on.

Projektin alussa käyttöliittymää suunniteltiin piirtämällä.



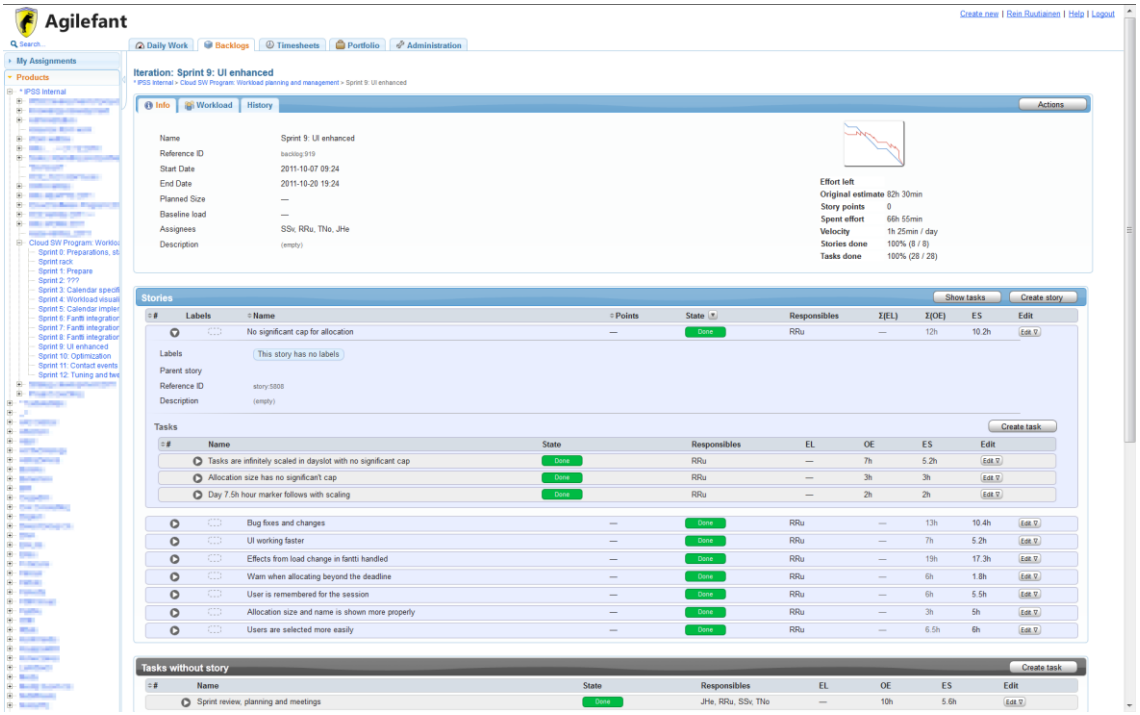
Kuvio 2. Käyttöliittymäsuunnitelman lopullinen versio.

Viimeinen versio käyttöliittymän piirustuksesta näkyy kuviossa 2. Tämän pohjalta lähti eteenpäin itse sovelluksen kehitystyö.

3 Projektinhallinta Agilefantilla

3.1 Työkuorman hallinta

Agilefant on suomalainen avoimen lähdekoodin ketteriä menetelmiä tukeva projektinhallintatyökalu [3].

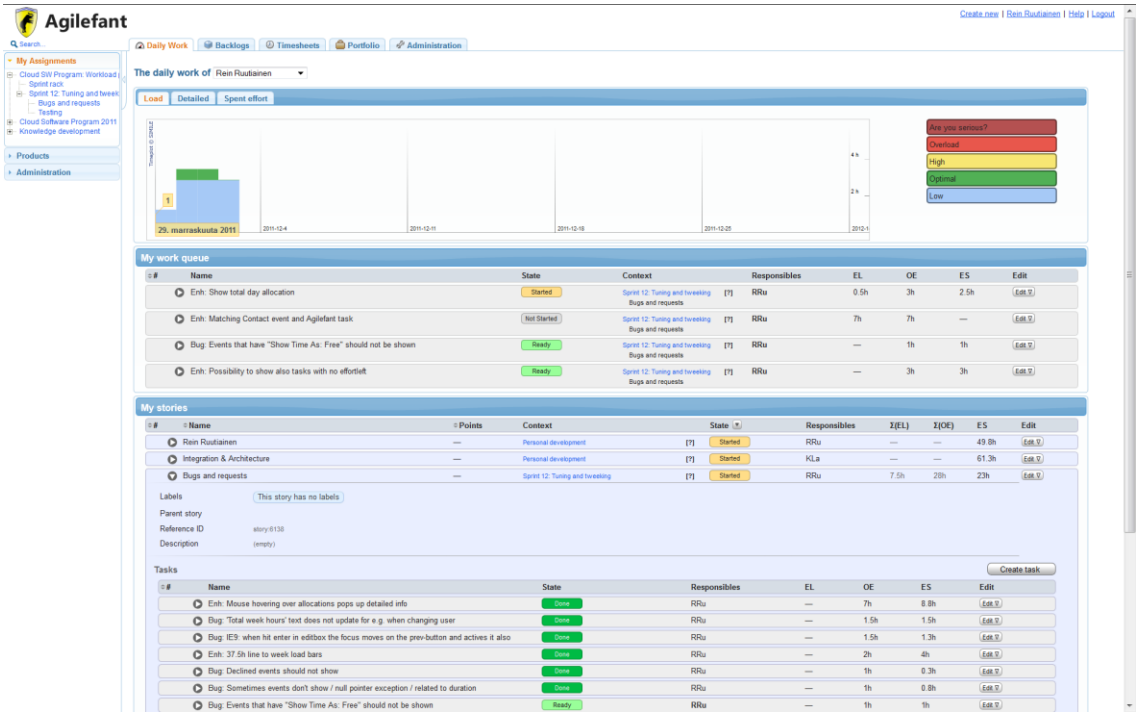


Kuvio 3. Esimerkkinä LoTo:n projektin yhdeksännen iteraation backlog.

Iteraation backlog, eli tuotteen työluettelo, joka sisältää kaikki tuotteelle määritetyt vaatimukset, näyttää listattuna, mitä tarinoita ja tehtäviä iteraatioon kuuluu. Kuviossa 3 alimpana kohdassa *Tasks without story* on nimensä mukaisesti tehtävät ilman tarinaa.

Tärkeää tietoa LoTo:n kannalta on muun muassa Agilefantin tietojen rakenne, eli miten on käsitelty iteraatiot, tarinat ja tehtävät ja näiden muodostama hierarkia. Agilefantin tietojen rakenne koostuu hierarkiassa ylimmällä tasolla tuotteista, joiden alla on projektit. Projektit muodostuvat taas iteraatioista ja iteraatiot muodostuvat käyttäjätarinoista, jotka vielä pilkkoutuvat tehtäviin. Iteraatioiden kohdalla voi olla suoraan myös tehtäviä.

Kuvion 3 vasemmassa reunassa on puurakenne, jossa ovat selattavat tuotteet, projektit ja iteraatiot.



Kuvio 4. Esimerkki työkuorman näkymästä Agilefantissa.

Kuviossa 4 on esitetty käyttäjän työtilanteen katsaus tarinoineen ja tehtävineen. Ylhäällä näkyy työkuorman jakauma visuaalisesti arvioituna tuleville viikoille. Keskellä on työjonolista, johon käyttäjä voi siirtää itselleen merkittyjä tehtäviä näkyviin, jos haluaa järjestellä, missä järjestyksessä tehtävänsä tekee. Alhaalla on lista kaikista käyttäjälle merkityistä tarinoista ja tehtävistä, jotka ovat jossain meneillään olevassa iteraatiossa.

Tehtävien kestolle voi määrittää ajallisia arvioita, joista käytetään nimitystä OE *Original estimate* eli alkuperäinen arvio tehtävän kestolle, kun tehtävä perustetaan. EL on *Effort left*, joka kertoo, kuinka paljon on tämänhetkinen arvio tehtävän jäljellä olevasta kestosta. ES *Effort spent* kertoo, kuinka paljon työtehtävälle on kirjattu tehtyä työaikaa. Nämä lyhenteet näkyvät kuviossa 4 tehtävien perässä. Näistä tiedoista tietoa *Effort left* käytettään vain LoTo:ssa.

Jäljellä olevan arvioidun työmäärän mukaan piirretään useamman tulevan viikon päivistä pylväsdiagrammeja, jotka ilmaisevat päiväkohtaista työmäärää ja ovat nähtävissä kuvion 4 yläosassa. Nämä päiväkohtaisen työmäärän kuvaajat piirretään automaatti-

sesti laskettuna tehtävistä, jotka on merkitty käyttäjän omalle vastuulle, jakaen näiden tehtävien jäljellä oleva työmäärä kuluvasta päivästä lähtien iteraation määräajan loppuun asti. Samalle päivälle tulevien eri tehtävien työmäärä summataan yhteen palkkiin. Agilefantissa on myös *Detailed*-näkymä työtaakan kuvaajasta, jossa näkee päiväkohtaisesti, mistä tehtävistä pylväs tarkemmin muodostuu. Päivittäinen työmäärä eli pylväiden korkeus mitoittuu sen mukaan, kuinka paljon on Agilefantin asetuksista määrittänyt itselleen arvioiduksi viikon työmääräksi.

3.2 Puutteet työkuorman hallinnassa

Merkittävä puute Agilefantissa on, että ei voi kirjata itse minä päivänä tekee mitäkin tehtävää. Vaikka Agilefantissa on mahdollisuus laatia itselleen priorisoitu työjono tehtävistä, niin työaikaa vievät myös erinäiset tapaamiset, jotka on merkitty iSteer Contactin kalenteriin. Työtehtävät tulisi suunnitella sopivasti tapaamisten ohelle, jolloin voisi huomioida paremmin, mitä tulevien viikkojen aikana oikeasti ehtisi tehdä ennen määräaikojen koittamista. Tähän perustuu tarve LoTo:lle.

4 Käytetty tekniikka

Tässä luvussa käydään läpi käytettyjä tekniikoita sekä arkkitehtuuria. Integraatio- ja tietokantaratkaisut käsitellään luvussa 5 ja ohjelman toiminnallisuus sekä tilannekohtaiset tekniset ratkaisut käydään läpi luvussa 6, jossa kerrotaan LoTo:n toiminnoista käyttöesimerkkien avulla.

LoTo on toteutettu verkko-ohjelmaksi ja ohjelmointitekniikkana on käytetty palvelinpuolella avoimen lähdekoodin Java Spring -ohjelmistokehystä ja tämän Spring MVC -moduulia Javalle [4]. Sovellus siis toteuttaa MVC-arkkitehtuuria. Käyttäjäpuoli on toteutettu JavaScriptin jQuery-kirjastolla [5]. Tietokantoja luetaan ja hallitaan Hibernate ORM -ohjelmistokehyksellä, jolla voi luoda tietokantataulujen rivejä olioiksi ja vastaavasti tallentaa näitä olioita tietokantatauluihin uusiksi riveiksi [6; 20].

4.1 Spring MVC

Spring helpottaa ohjelmointia muun muassa tuomalla monia automatisoivia piirteitä Javaan ja pidemmän päälle helpottaa koodin ylläpidettävyyttä. Tässä käydään läpi vain niitä ominaisuuksia, mitä LoTo:ssa on käytetty.

Spring käyttää keskeisesti *Dependency Injection* -suunnittelumallia. Springissä tällä voidaan mahdollistaa XML-tiedostoissa asetettujen tai annotaatioilla asetettujen olioiden syöttäminen suoraan attribuutteihin. Näitä olioita kutsutaan *Spring beaneiksi* [16]. [7; 20, s. 231–234.]

Dependency Injection -suunnittelumallissa ajatuksena on tuoda luokan olioattribuuttien instanssit ulkoapäin, kuten rakentajan välittämänä. Täten voi sijoittaa korkeammalta tasolta haluamansa olion, joka toteuttaa vaaditun rajapinnan. [7; 20, s. 231–234.]

Tämä helpottaa testien kirjoittamista, kun voi käyttää mock-up -luokkia eli tekaistuja luokkia. Mock-up -luokat eivät ole oikeita luokkia, mutta näyttäytyvät sellaisina vain rajapinnaltaan. Luokka, josta mock-up tehdään, sisältää samat metodit kuin alkuperäinen luokka, mutta näillä ei ole toiminnallisuutta itsessään, vaan testaaja itse määrittää

mitä nämä metodit tuottaa. Näin testaaja voi kokeilla, mitä seurauksia on eri palautusarvoilla.

```
@Controller
public class MyController {

@Autowired private MyClassBo myClassBoBean;

@RequestMapping(value="/pathToMyAction", method=RequestMethod.POST)
public @ResponseBody void myAction(@RequestParam String text, @RequestParam int
number, HttpSession session)
{
  String sessionText = (String)session.getAttribute("sessionText");

  if (text.equals("")) {
    throw new IllegalArgumentException("text is empty");
  }
  if (number < 0) {
    throw new IllegalArgumentException("number is zero or negative");
  }
  if (sessionText == null) {
    throw new IllegalArgumentException("sessionText is null");
  }

  myClassBoBean.doBusiness(text, number, sessionText);
}
}
```

Koodilistaus 1. Esimerkki Spring MVC:n käsittelijästä.

Koodilistaus 1:ssä käsittelijä-luokalle asetetaan *@Controller*-annotaatio, joka mahdollistaa monia Spring MVC -moduulin käsittelijäkohtaisia toimintoja. Näin ei tarvitse käyttää *doPost*- tai *doGet*-metodeja. Metodit kuvataan toimimaan eri osoitteita vasten *@RequestMapping*-annotaatiolla. HTTP-pyynnön parametrit saadaan annotoimalla metodin parametrit *@RequestParam*-annotaatiolla eikä tarvitse käsitellä *HTTPRequest-Servlettiä*. Liittyen *Dependency Injection* -tekniikkaan *@Autowired*-annotaatiolla saa tuotua beanit attribuutteihin, jotka ovat LoTo:n tapauksessa määritettyinä XML-tiedostoihin. [8; 9.]

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

 <!-- MyClass Business Object -->
 <bean id="myClassBoBean" class="my.package.path.bo.impl.MyClassBoImplementation">
   <property name="myClassDao" ref="myClassDaoBean" />
 </bean>

  <!-- MyClass Data Access Object -->
 <bean id="myClassDaoBean" class="my.package.path.dao.impl.MyClassDaoImplementation">
   <property name="sessionFactory" ref="mySessionFactoryBean"></property>
 </bean>

</beans>
```

Koodilistaus 2. Esimerkki Spring beanin XML-asetuksista.

Koodilistaus 2:n esimerkissä *myClassBoBean* alustetaan luokalla *MyClassBoImplementation*. Tällä luokalla on attribuutti *myClassDao*, jonka arvoksi asetetaan *myClassDaoBean*, joka on luokkaa *MyClassDaoImplementation*. DAO:n beanissa on attribuutti *sessionFactory*, jonka arvoa edustaa bean nimeltä *mySessionFactoryBean*. Tämän beanin asettamista ei ole esimerkissä esitetty, mutta *sessionFactory*-attribuutti edustaisi Hibernatelle asetettua tietokantayhteyttä. BO- ja DAO-luokat käsitellään luvussa 5.

4.2 Käyttöliittymän tekniikka

Käyttäjäpuolella toiminnot on tehty JavaScriptin avoimen lähdekoodin jQuery-kirjastolla, joka tarjoaa normaalia JavaScriptia tehokkaamman tavan hallita DOM-elementtejä, CSS-manipulointia, AJAX-kutsuja, tapahtumia ja paljon muuta. [5.]

Hyvin suuri osa koko LoTo-projektia on ollut käyttöliittymätoimintojen tekemistä, ja täten jQueryllä toteutettua koodia on kertynytkin suurin osa. Kaikki yhteydenotot käyttäjäpuolelta palvelinpuolelle käydään jQueryn AJAX-kutsuilla, sillä käyttöliittymä toimii näin sulavammin, kun jokaisen pienen muutoksen jälkeen ei päivitetä koko sivua. Käyttäjäpuoli lähettää pyyntöjä, joiden perusteella käsittelijän kautta joko haetaan tietokannasta tietoa tai lähetettyä tietoa vahvistetaan. Tietokannasta haettua tietoa voidaan vielä muokata erilaiseen haluttuun muotoon, ennen kuin se lähetetään JSON-muodossa käyttäjäpuolelle.

LoTo:ssa on vain yksi käsittelijä-luokka sekä näkymänä toimiva HTML-sivu, joka toimii pohjana sivuston ulkoasulle. Muita näkymiä ei LoTo:ssa ole. Näkymään kirjoitettavia

palvelinpuolen toimintoja, kuten skriplettejä tai JSTL:ää ei ole käytetty. Näkymään määriteltyyn HTML:ään lukeutuu esimerkiksi sivuston yleinen ulkoasu, kuten tausta, painikkeet ja kaikki mitä voi välittömästi näyttää käyttöliittymässä eikä vaadi dynaamista luontia.

On olemassa paljon erilaisia valmiiksi tehtyjä laajennuksia jQuerylle, ja usein ne vielä ovat vapaasti käytettävissä GPL- tai MIT-lisenssien puitteissa, kuten vaikka erilaiset valmiit kalenterit tai puunäkymät. Työssä päädyttiin käyttämään jQuery-pääkirjaston lisäksi jQuery UI -kirjastoa sekä jQuery Tools -laajennuskokoelmasta Tooltip-laajennusta, jolla voi helposti luoda työkaluvihjeitä. [10; 11.]

Näistä JavaScript kirjastoista jQuery UI -kirjasto, jonka nimessä UI eli *User Interface* tarkoittaa käyttöliittymää, tuo jQueryyn mahdollisuuden helposti käsitellä ja tehdä visuaalisia toimintoja, efektejä ja tapahtumia. Esimerkiksi mahdollistaa helposti HTML elementtien liikuttamisen hiirellä ja saa elementin koon muuttumaan haluamallaan tavalla. Erilaisista efekteistä löytyy esimerkiksi elementin katoaminen tai näkyviin tuleminen rauhallisesti häivyttämällä. Elementtejä voi myös elävöittää eri tavoin, kuten esimerkiksi saamalla pyörimään, muuttamaan muotoaan ja liikkumaan. Löytyy valmis pohja dialogi-ikkunalle, minikalenterille ja monille muille käyttöliittymä toiminnoille. [10.]

5 Tietolähteiden integraatio

LoTo:n tietokantatoiminnot voisi jakaa kolmeen kohteeseen, vaikka teknisesti LoTo:ssa käytetään kahta eri tietolähdettä. On tarpeen lukea Agilefantin tietoja ja toisena on tarpeen tietää myös IPSS iSteer Contact CRM -ohjelmiston kalenterintapahtumat. Kolmanneksi täytyy tallentaa ja ladata omat työsuunnitelmat.

5.1 Käytetyt suunnittelumallit

Projekti koostuu rakenteeltaan yhden käsittelijän ja näkymän lisäksi myös useista DTO-, BO- ja DAO-luokista. Kaikki kolme ovat suunnittelumalleja, joita käytetään jakamaan eri toiminnot toisistaan. Näiden käyttäminen on yksi tapa pitää käsittelijät sekä tietokantahaut siistimpinä, kun koodit on erotettu toisistaan eri kerroksiin.

Hibernatella pääsee mukavalla tavalla käsiksi tietokantatauluihin luomalla DTO eli *Data Transfer Object* -malliluokkia, jotka vastaavat attribuuteiltaan haluttuja tietokantatauluja. DTO-malliluokat ovat tietokantataulurivien olio-vastineita. Näillä olioilla on attribuutteina samat tiedot, kuin vastaavilla tauluilla tietokannassa on sarakkeissa. [6; 20, s. 20–21.]

Hibernatella voi palauttaa tietokantahakujen tulokset suoraan DTO-malliluokkina. Yhteydet voi asettaa DTO-malliluokkien ja tietokantataulujen välille Hibernate XML -asetustiedostoilla tai sitten uudempia ominaisuuksia käyttäen voi annotaatioilla kirjoittaa suoraan DTO-malliluokkiin tarvittavat tiedot. LoTo:ssa DTO-malliluokat asetetaan annotaatioilla. [12.]

BO-luokat ovat *Business Object* -luokkia tai toiselta nimeltään palveluluokkia, joiden tehtävänä on hoitaa logiikka, joka koskee tietokantahakuja. [13.]

DAO eli *Data Access Object* -luokissa tehdään itse tietokantahaut, joiden tulisi olla yksinkertaisia eikä sisältää logiikkaa, mikä tulisi hoitaa BO-luokissa. DAO-luokkia ei käytetä suoraan käsittelijästä vaan DAO-luokka kuuluu attribuuttina BO-luokalle, jossa sitä käytetään tarvittuihin tietokantahakuihin. Käsittelijässä tehdään tietokantoja koskevat kutsut BO-luokalla. [14.]

Jokaista taulua kohden on omat DTO-, BO- ja DAO-luokat. Esimerkiksi LoTo-projektissa on olemassa taulu nimeltä *allocations*, jota kohden on olemassa *AllocationsDto-*, *AllocationsBo-* ja *AllocationsDao*-luokat.

Hibernatessa hakuja tehdään SQL:ää muistuttavalla HQL-kielellä tai käyttäen Hibernate Criteria -rajapintaa, jolla voi metodein koostaa koko haun. Myös SQL-lausekkeilla voi tehdä hakuja. [20, s. 199–200.]

5.2 Agilefant-integraatio

Ennen kuin LoTo:ssa voi suunnitella mitään, on luettava Agilefantin tietoja ja nämä luetaan suoraan Agilefantin tietokannasta Hibernatea käyttäen. Agilefantin tietokantaan luotiin myös uusi käyttäjä LoTo:n käytettäväksi, jolla on vain lukuoikeus tietoihin.

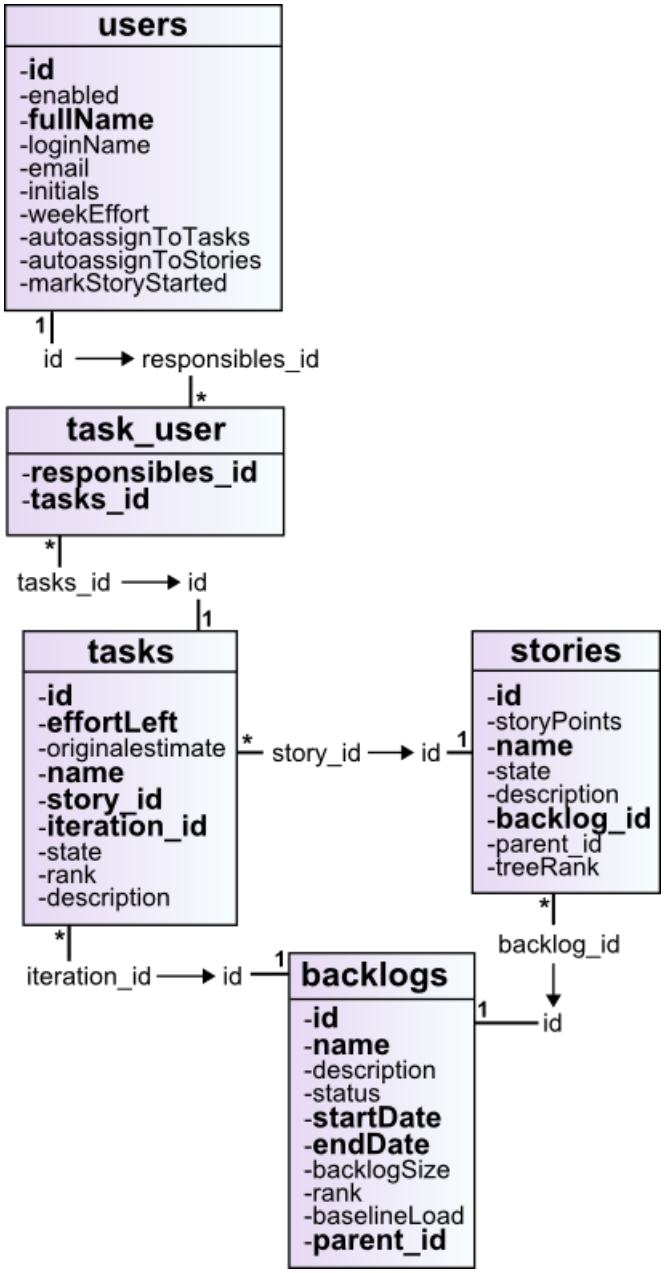
Taulukko 1. Tarvittavat Agilefantin taulut ja niitä vastaavat DTO-malliluokat.

Agilefantin taulu	**Hibernate annotuitu DTO**	**Kuvaus**
users	UsersDto	Käyttäjän tiedot, kuten nimi.
tasks	TasksDto	Tehtävän tiedot, kuten tehtävän nimi ja arvioidun työmäärän kesto sekä ID mihin tarinaan kuuluu.
stories	StoriesDto	Tarinan tiedot, kuten nimi ja ID mihin backlogiin kuuluu.
backlogs	BacklogsDto	Edustaa iteraatiota, projektia sekä tuotetta.

Agilefantissa on parikymmentä taulua, mutta niistä tarvitaan vain neljää varsinaista sekä yksi välitaulu **task_user** yhdistämään **users-** ja **tasks**-taulut.

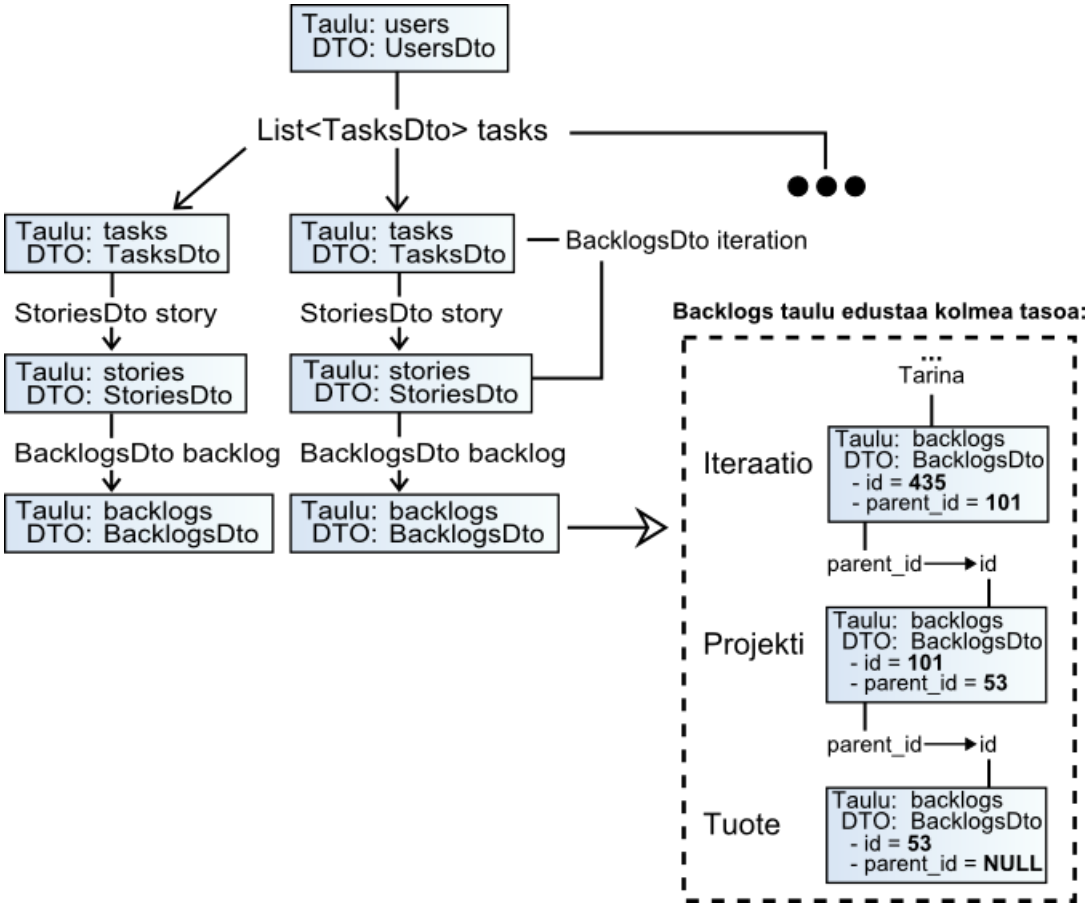
Backlogs-taulu edustaa iteraatioita, projekteja sekä tuotteita, sillä kaikki nämä kolme niin sanottua eri tasoa koostuvat samanlaisista tiedoista. Tässä taulussa on *parent_id*-attribuutti, jolla viitataan saman taulun toisen rivin tunnisteeseen, jotta saataisiin yhte-

ys niin sanotusti korkeampaan tasoon, kuten iteraatiosta projektiin ja projektista tuotteeseen. Tuotetasolla eli ylimmällä tasolla *parent_id* on tyhjä.



Kuvio 5. Agilefantin tietokannasta tarvittavat taulut.

LoTo:ssa käytetään ainoastaan kuvion 5 taulujen lihavoituja attribuutteja. Näistä tauluista luodaan siis DTO-malliluokkia eli luokkia, joilla on näiden taulujen sarakkeet attribuutteina sekä get- ja set-metodit.



Kuvio 6. Agilefantin tauluja edustavien DTO-malliluokkien keskeiset suhteet.

Hibernatella voi asettaa myös taulujen välisiä yhteyksiä, ja kuvio 6 antaa tästä yleiskuvan. LoTo:ssa DTO-malliluokat on asetettu siten, että Hibernatella hakiessa UsersDto-luokalla on listana kaikki tämän kyseisen käyttäjän tehtävät. Tehtävillä on attribuuttina isäntätarina tai -iteraatio. Tehtävä voi olla Agilefantissa merkittynä tarinaan tai vaihtoehtoisesti suoraan iteraatioon. Suoraan iteraation alla olevat tehtävät on Agilefantissa merkitty *Tasks without stories* -tehtäviksi.

LoTo:ssa tarinat ovat vain iteraatioissa, vaikka Agilefantissa tarinat voivat olla myös tuote- tai projektitasolla. LoTo:ssa näytetään ainoastaan iteraatioiden tarinat, sillä Agilefantissa näyttää mahdolliselta luoda tehtäviä vain näille tarinoille. Tehtävät ovat olennaista tietoa, sillä vain niitä voi käyttäjä sijoittaa LoTo:ssa työkuormakalenteriin.

Hibernate-annotaatioita käyttäessä DTO-malliluokille asetetaan luokkamäärityksen eteen annotaatio *@Entity* merkitsemään, että kyseessä on tallennettava olio ja lisäksi

@Table-annotaatio merkitsemään, mitä tietokantataulua tämä tallennettava olio vastaa. [15.]

```
@Entity
@Table(name = "users")
public class UsersDto implements Serializable {

private static final long serialVersionUID = 2806972957173082659L;
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
@Column(name = "id") private int id;
@Column(name = "enabled") private short enabled;
@Column(name = "fullName") private String fullName;
@Column(name = "loginName") private String loginName;
@Column(name = "email") private String email;
@Column(name = "initials") private String initials;
@Column(name = "weekEffort") private Long weekEffort;
@Column(name = "autoassignToTasks") private Boolean autoassignToTasks;
@Column(name = "autoassignToStories") private Boolean autoassignToStories;
@Column(name = "markStoryStarted") private Integer markStoryStarted;
@OneToMany(fetch=FetchType.EAGER)
@JoinTable(
name = "task_user",
joinColumns = @JoinColumn(name="responsibles_id"),
inverseJoinColumns = @JoinColumn(name="tasks_id")
)
private List<TasksDto> tasks;

// getterit ja setterit normaalisti

}
```

Koodilistaus 3. DTO-malliluokka users-taulusta.

Koodilistauksen 3 esimerkissä attribuuteilla on *@Column*-annotaatio. Tämä yhdistää attribuutin taulun oikeaan sarakkeeseen. Taulun *id* on merkitty esimerkin mukaisesti omilla annotaatioilla tämän lisäksi. Attribuutti-kohtaiset annotaatiot voi sijoittaa myös tämän tilalta get-metodeihin. [15.]

Hibernatella voi asettaa DTO-malliluokkaan taulujen välisiä suhteita lisäämällä attribuutin, joka annotoidaan vastaamaan jotain muuta arvoa tietokannasta kuin vain oman taulunsa saraketta. LoTo:ssa Hibernatella on toteutettu useampi yhdestä moneen tai monesta yhteen yhteys.

Esimerkissä attribuutille *tasks* toteutetaan annotaatioilla *@OneToMany*, *@JoinTable* ja *@JoinColumn* käyttäen yhdestä moneen yhteys, merkitsemällä välitaulun **task_user** yhdistävät ID-arvot liittämään **tasks**-taulu **users**-tauluun. Attribuutin *tasks* ollessa myös tyyppiä *List<TasksDto>* saadaan Hibernatella käyttäjiä hakiessa tähän attribuuttiin kaikki tälle käyttäjälle merkityt tehtävät. [15.]

```
@Entity
@Table(name = "tasks")
public class TasksDto implements Serializable {

private static final long serialVersionUID = -7719405387291225280L;
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
@Column(name = "id") private int id;
@Column(name = "effortLeft") private Long effortLeft;
@Column(name = "originalestimate") private Long originalestimate;
@Column(name = "name") private String name;
@Column(name = "state") private int state;
@Column(name = "rank") private int rank;
@Column(name = "description") private String description;
@Column(name = "iteration_id") private Integer iterationId;
@Column(name = "story_id") private Integer storyId;
@ManyToOne
@JoinColumn(name = "story_id", insertable = false, updatable = false)
private StoriesDto story;
@ManyToOne
@JoinColumn(name = "iteration_id", insertable = false, updatable = false)
private BacklogsDto backlog;

// getterit ja setterit normaalisti

}
```

Koodilistaus 4. DTO-malliluokka tasks-taulusta.

Tehtävien eli **tasks**-taulun DTO-malliluokka, kuten koodilistaus 4 näyttää, sisältää kaksi monesta yhteen annotoitua attribuuttia, jotka edustavat tehtävän isäntätarinaa ja isäntäiteraatioota. Agilefantissa näistä voi olla voimassa vain yksi kerrallaan ja toinen on silloin *null*. Tämä määrittää Agilefantissa kuuluuko tehtävä tarinaan vai onko tehtävä iteraatiotasolla eli samalla niin kutsuttu *Task without story*. LoTo:n koodissa tätä myös seurataan saman asian suhteen, sillä puunäkymää rakentaessa tulee tietää, mille tasolle tehtävä on merkitty.

```
@Entity
@Table(name = "stories")
public class StoriesDto implements Serializable {

private static final long serialVersionUID = -1131324579468978577L;
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
@Column(name = "id") private int id;
@Column(name = "storyPoints") private Integer storyPoints;
@Column(name = "name") private String name;
@Column(name = "state") private int state;
@Column(name = "description") private String description;
@Column(name = "backlog_id") private int backlog_id;
@Column(name = "parent_id") private Integer parent_id;
@Column(name = "treeRank") private int treeRank;
@ManyToOne
@JoinColumn(name = "backlog_id", insertable = false, updatable = false)
private BacklogsDto backlog;

// getterit ja setterit normaalisti

}
```

Koodilistaus 5. DTO-malliluokka stories-taulusta.

Koodilistaus 5:ssa käsitellään **stories**-taulu. Agilefantin luetuista tauluista **stories** on hyvin samankaltainen kuin **tasks** monesta yhteen suhteella, mutta **tasks**-taulu on vain yhteydessä isäntäänsä **backlogs**-taulussa.

```
@Entity
@Table(name = "backlogs")
public class BacklogsDto implements Serializable {

private static final long serialVersionUID = -2463611715014899647L;
@Column(name = "backlogtype") private String backlogtype;
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
@Column(name = "id") private int id;
@Column(name = "name") private String name;
@Column(name = "description") private String description;
@Column(name = "status") private Integer status;
@Column(name = "startDate") private Date startDate;
@Column(name = "endDate") private Date endDate;
@Column(name = "backlogSize") private Long backlogSize;
@Column(name = "rank") private Integer rank;
@Column(name = "baselineLoad") private Long baselineLoad;
@Column(name = "parent_id") private Integer parent_id;
@ManyToOne
@JoinColumn(name = "parent_id", insertable = false, updatable = false)
private BacklogsDto parent;

// getterit ja setterit normaalisti

}
```

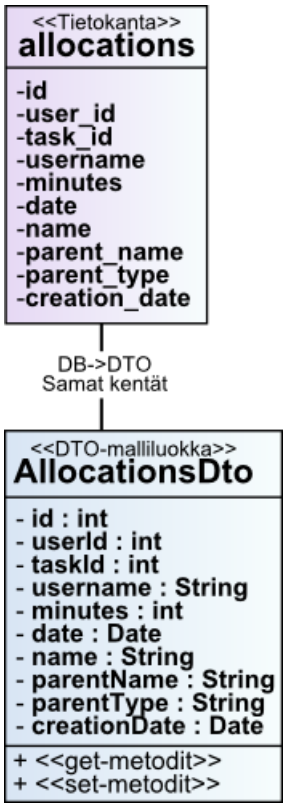
Koodilistaus 6. DTO-malliluokka backlogs-taulusta.

Koodilistaus 6 esittää **backlogs**-taulua vastaavan DTO:n. Tämä omistaa monesta yhteen -yhteyden, ja tässä tulee huomioida, että **backlogs**-taulu on yhteydessä itseensä. Taulussa attribuutti *parent_id* viittaa saman taulun toiseen riviin, joka edustaa isäntää.

Taulu **backlogs** edustaa kolmea eri tasoa, jotka ovat matalimmasta korkeimpaan iteraatio, projekti ja tuote. Esimerkiksi iteraatiotasolla *parent_id* viittaisi riviin, joka edustaisi projektia ja projektitason *parent_id* viittaisi saman taulun toiseen riviin, joka edustaisi tuotetta. Tuotetason ollessa korkein on *parent_id* arvoltaan *null*. Taulussa *backlog-type* kertoo mitä tasoa rivi edustaa.

5.3 Työsuunnitelman tallennus

Työviikot suunnitellaan sijoittamalla Agilefantin tehtäviä LoTo:n kalenteriin, niinpä kaikki sijoitukset tulee myös tallentaa myöhempää käyttöä varten. LoTo:lle on tehty oma MySQL-tietokanta, jossa kaikki tallennukset tehdään tauluun nimeltä **allocations**.



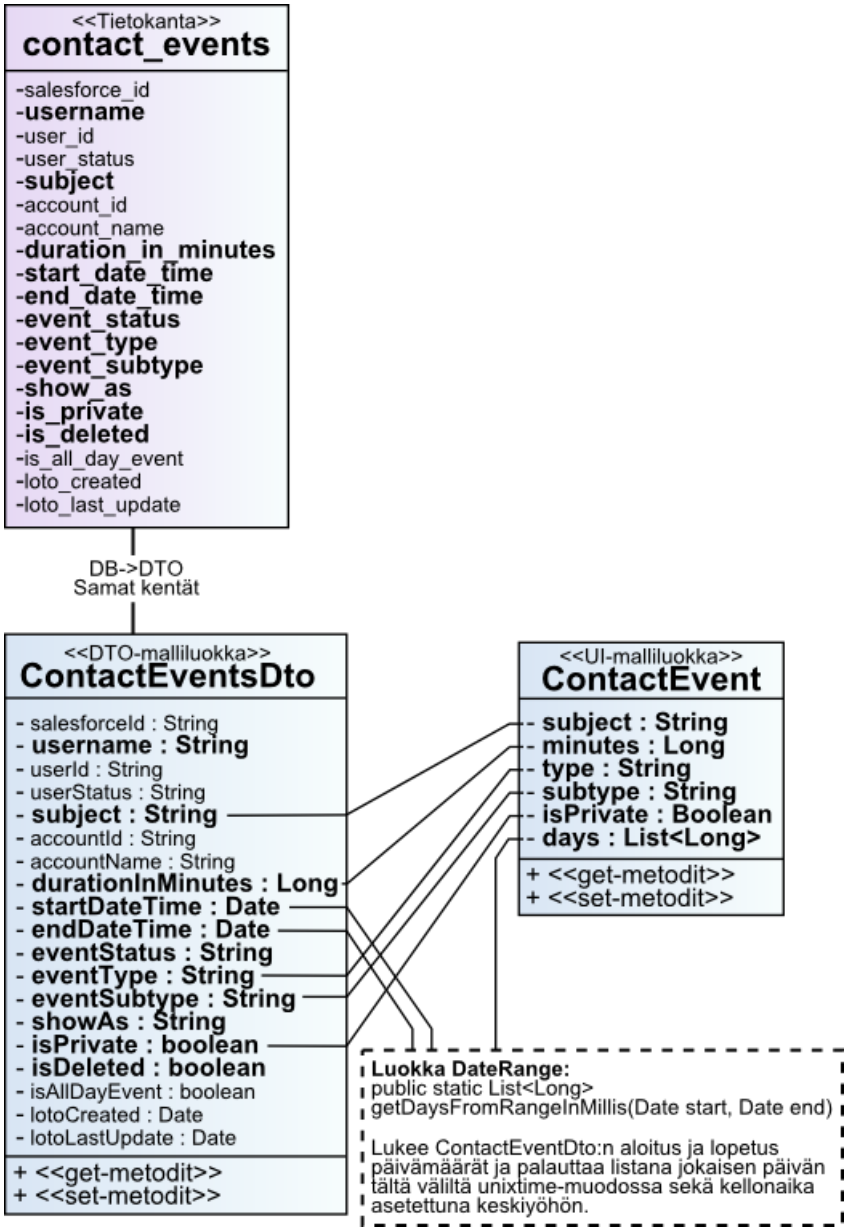
Kuvio 7. Allocations-taulu ja DTO-luokka.

Kaikki tallennukset tapahtuvat Ajaxin välityksellä ja jokaisen kalenteriin sijoituksen yhteydessä. Tallennus tehdään sijoitettaessa uutta Agilefantin tehtävää LoTo:n kalenteriin, siirtäessä jo kalenterissa olevaa tehtävää toiselle päivälle tai muokatessa kalenterissa jo olevan tehtävä sijoituksen työmäärää.

IPSS iSteer Contactin kalenteritapahtumia ei tallenneta tai muokata LoTo:ssa. Ne vain ladataan suoraan kalenterinäkymään. Seuraavassa luvun osiossa 5.4 käsitellään asiaa.

5.4 IPSS iSteer Contact -integraatio

Contactin tapahtumat sijaitsevat yhdessä taulussa LoTo:n tietokannassa, johon ne tuodaan Contactin omasta lähteestä nykyään viidentoista minuutin välein, eli LoTo:n Contactin tapahtumatiedot ovat vain kopioita alkuperäisestä. Näihin tietoihin ei tehdä muutoksia LoTo:n kautta, sillä LoTo:ssa nämä tiedot ovat vain lukukäytössä.



Kuvio 8. Contactin tapahtuma-taulu, DTO- ja UI-malli.

Tapahtumien DTO-malliluokalla on paljon attribuutteja, joita ei näkymän puolella tarvita. Tämän vuoksi näkymää varten on oma malliluokka ContactEvent. Tämän malliluokan attribuutti *days* ei löydy suoraan DTO:sta ja on tehty edustamaan DTO:n *startDateTime* ja *endDateTime* välin päiviä unixtime-muodossa.

```
public static List<Long> getDaysFromRangeInMillis(Date start, Date end) {
   List<Long> allDays = new ArrayList<Long>();
   long daysBetween = ((end.getTime() - start.getTime()) / 86400000) + 1;
   Calendar calendar = Calendar.getInstance();
   calendar.setTime(start);
   int dayOfWeek;
   for (int day = 0; day < daysBetween; day++) {
      dayOfWeek = calendar.get(Calendar.DAY_OF_WEEK);
      if (!(dayOfWeek == Calendar.SATURDAY || dayOfWeek == Calendar.SUNDAY)) {
         allDays.add(calendar.getTimeInMillis());
      }
      calendar.add(Calendar.DAY_OF_MONTH, 1);
   }
   return allDays;
}
```

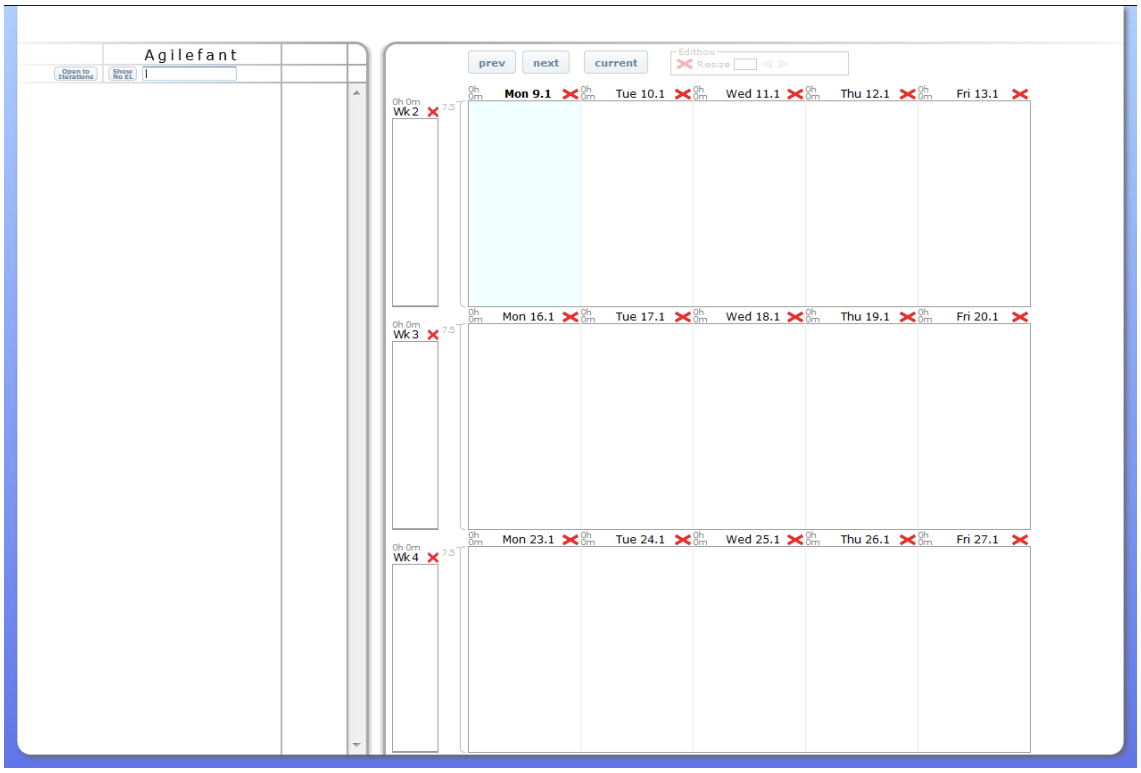
Koodilistaus 7. Kahden päivämäärän välisen ajan muuttaminen unixtime listaksi.

Contactissa on merkitty tapahtumalle aloitus- ja loppupäivämäärä. LoTo:ssa käytetään tämän aikavälin jokaisen päivän keskiyötä unixtime-muodossa merkitsemään tapahtuman päiviä. Koodilistaus 7 esittää tämän muutoksen.

6 Toteutus

6.1 Ulkoasu

Projektissa luotiin suunnitelmissa hahmotetun ulkoasun mukainen HTML-sivu, joka on kuviossa 9. Ulkoasu on toteutettu CSS:llä. Ainoastaan painikkeissa käytetään myös kuvia.



Kuvio 9. LoTo:n ulkoasu, kun käyttäjää ei ole haettu.

Lopullinen ulkoasu on varsin samankaltainen kuin suunnitelmiin piirretty. Erona on näkyvien viikkojen määrä, neljän sijaan kolme, sillä päivämerkinnöille tarvitaan enemmän tilaa, tekemättä kuitenkaan sivusta liian korkeaa, jotta se näkyisi useimmilla näytöillä kokonaan. Toisena myös suunnitelmien piirustuksessa oli laitettu neljä valintakohtaa Agilefantin sisällön suodattamiselle, mutta kaiken tarpeellisen saa etsittyä käyttäjänimen mukaan yhdestä kohtaa.

Kuviossa 9 ei ole vielä näkyvillä yhtäkään kalenterimerkintää tai aikaisemmin mainittua puunäkymää Agilefantin tehtävistä, sillä käyttäjää ei ole vielä haettu. Näkymä siis näyttää tältä, kun sivustolle saapuu ilman, että evästettä on vielä luotu aikaisemmasta Lo-

To:n käytöstä. LoTo käyttää evästettä tallentamaan viimeksi haetun käyttäjänimen. Tämän avulla käyttäjätiedot haetaan takaisin sivulle palattaessa.

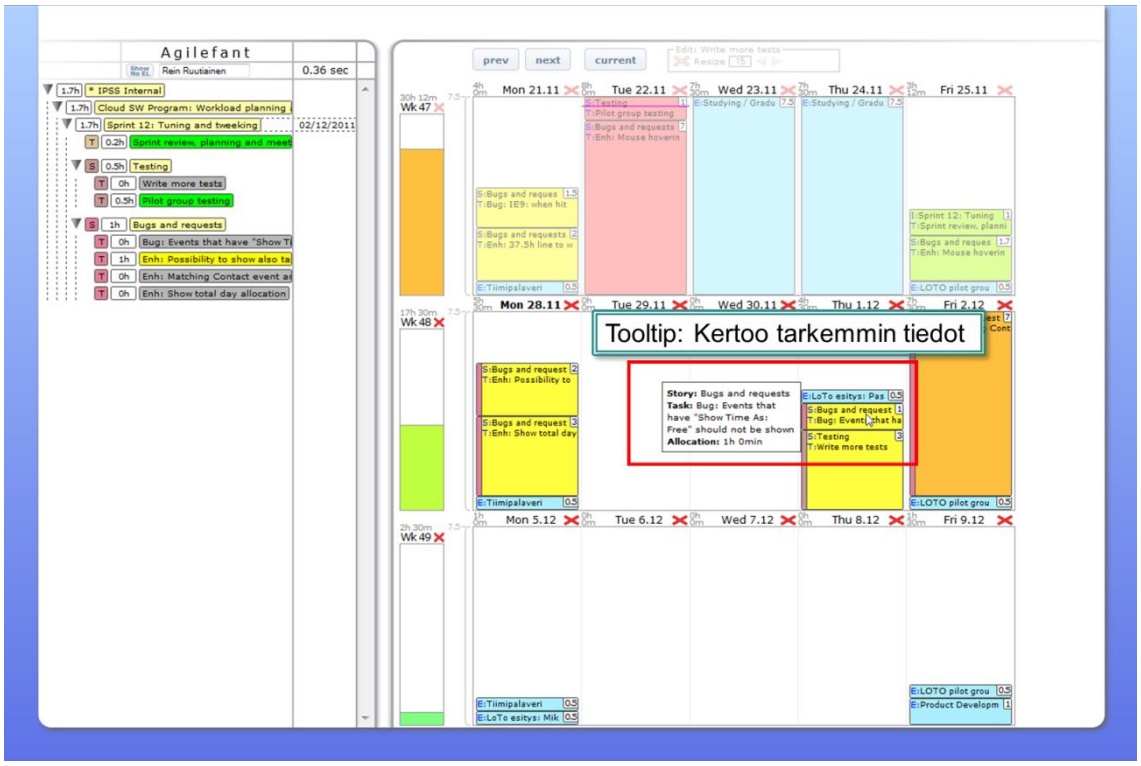
6.2 Käyttäjätietojen hakeminen

Käyttäjätietojen hakeminen Agilefantista on toteutettu tekstihakukentällä, jossa on tukena käyttäjänimien automaattinen täydennys.

Kuvio 10. Käyttäjätietojen hakemisessa käytetään automaattista täydennystä.

LoTo toimii vain IPSS:n sisäverkossa, ja siksi tunnistautumista ei ole. Käyttäjä haetaan kirjoittamalla se kuvion 10 osoittamalle paikalle. Käyttäjähaussa apuna toimii automaattinen täydennys, joka hakee käyttäjänimet Agilefantin tietokannasta, joista löytyy hakukenttään kirjoitettu teksti. Automaattinen täydentäminen hakee AJAXilla listan käyttäjistä heti, kun on kirjoittanut vähintään kaksi merkkiä. Haku suojataan erikoismerkkien vaikutuksilta. Automaattinen täydennys on jQuery UI:n tarjoama ominaisuus. Käyttäjä voidaan valita luettelosta hiirellä, näppäimistön nuolilla tai kirjoittamalla kokonaan. Ilman hiirtä valinta tapahtuu enter-näppäimellä.

Ylälaidan painikkeista voi kalenteriviikon vaihtaa viikko kerrallaan haluamaansa suuntaan ja voi palata meneillään olevan viikon kohdalle, jolloin kuluva viikko näkyy ylimpänä viikkona.



Kuvio 11. Esimerkissä on haettu henkilötiedot.

Kuviossa 11 nimeä vastaavat tiedot on haettu Agilefantista sekä IPSS iSteer Contactin tapahtumien tiedoista. Agilefantin tiedoista muodostetaan puunäkymä.

Puunäkymä rakennetaan Agilefantista noudettujen tietojen perusteella. Työkuormakalenterin merkinnät haetaan LoTo:n omasta tietokannasta, jonne kaikki käyttäjäkohtaiset LoTo-merkinnät on tallennettu. LoTo:n tietokannassa on käyttäjäkohtaiset Agilefantin tehtävien merkinnät sekä iSteer Contactin kalenteritapahtumat. Contactin tiedot tuodaan LoTo:n tietokantaan muualta viidentoista minuutin välein. Näitä tietoja ei muokata kuitenkaan LoTo:ssa ollenkaan, vaan niitä ainoastaan luetaan.

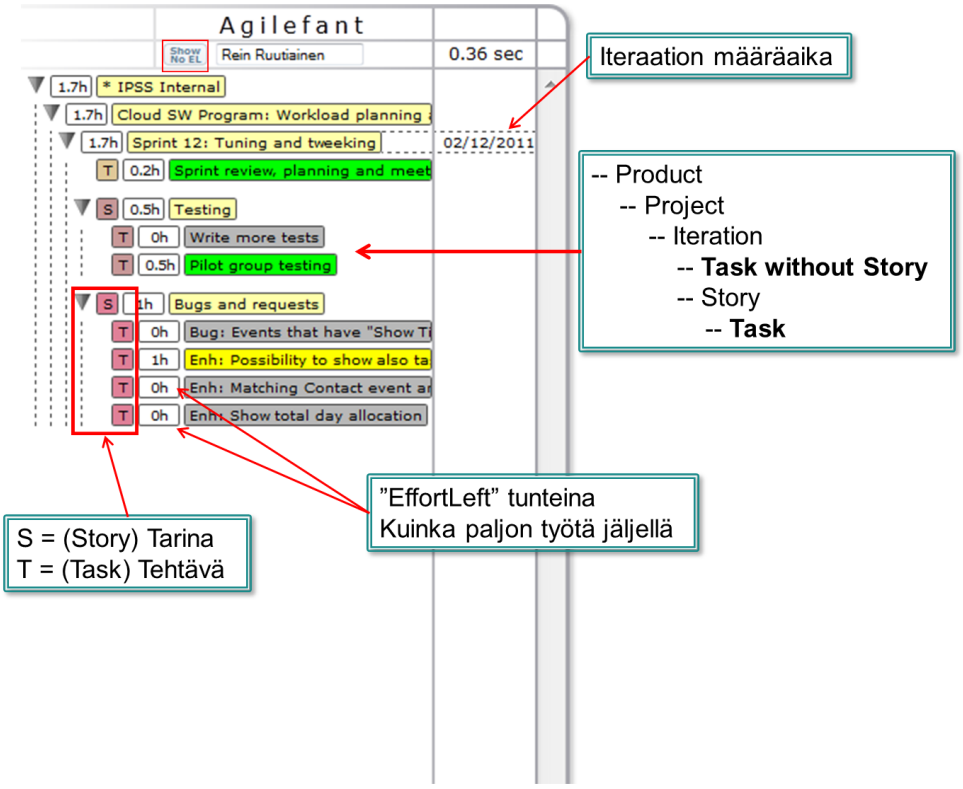
Kuvion 11 tapauksessa kalenterinäkymässä ylimmän viikon merkinnät ovat haaleamman värisiä kuin kahden seuraavan viikon siitä syystä, että nämä päivät ovat menneitä päiviä. Kaikkien menneiden päivien merkinnät haalennetaan, jotta luotaisiin ero tuleviin päiviin. Kuluvassa päivässä on päivämäärän teksti lihavoitu. Merkintöjä työkuorma-

kalenteriin voi tehdä Agilefantin tehtävistä vain kuluvalle tai tuleville päiville. Menneet päivät jäävät historiaan eikä niitä voi enää muokata tai poistaa eikä uusia merkintöjä voi näille päiville sijoittaa.

Kuviossa 11 on myös nähtävinä työkaluvihje, joka kertoo tarkemmin tehtävän tai Contact-tapahtuman tiedot, sillä jossain merkinnöissä nimi on pitempi kuin kentän leveys.

6.3 Puunäkymän muodostaminen

Kuviossa 12 on esimerkki siitä, miltä puunäkymä näyttää. Puunäkymä rakentuu saman tien, kun käyttäjä on haettu.

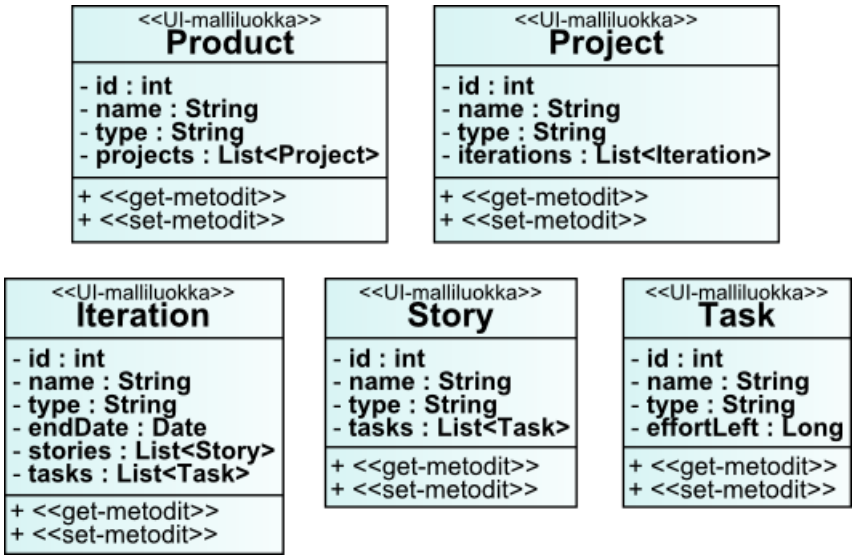


Kuvio 12. Puunäkymä, joka on rakennettu Agilefantin tiedoista.

Puunäkymä toteutettiin jQueryllä ja CSS:llä. Hiirellä nuolen kuvaa painamalla piiloutuvat tai paljastuvat sisemmät elementit ja näkee esimerkiksi mitä tehtäviä on tarinan alla.

Kaikki puunäkymään tuleva tieto haetaan kerralla kokonaan AJAXia käyttäen, kun valitsee halutun käyttäjän. Vaihtoehtoinen tapa olisi voinut olla, että haetaan tietoa sitä mukaa, kuin puunäkymän solmuja avaa, mutta kokonaisdatamäärä ei ole käytännössä koskaan niin suuri, että olisi liian raskasta hakea koko puu kerralla.

Palvelinpuolella tietokannasta noudettu Agilefantin data muokataan vastaamaan puunäkymän viittä eri tasoa käyttäen Javan listoja. Tasot ovat tuote, projekti, iteraatio, tarina ja tehtävä. Jokaiselle tasolle on oma malliluokka näkymää varten, jotta ei palautettaisi tietokannasta noudettuja DTO-luokkia, joissa on paljon ylimääräistä tietoa sekä päivämäärät muutetaan unixtime-muotoon, kuten aikaisemmin on mainittu. Kuviossa 13 on näkymässä käytettäviä UI-malliluokkien UML-mallinnukset.



Kuvio 13. LoTo:n UI-malliluokat.

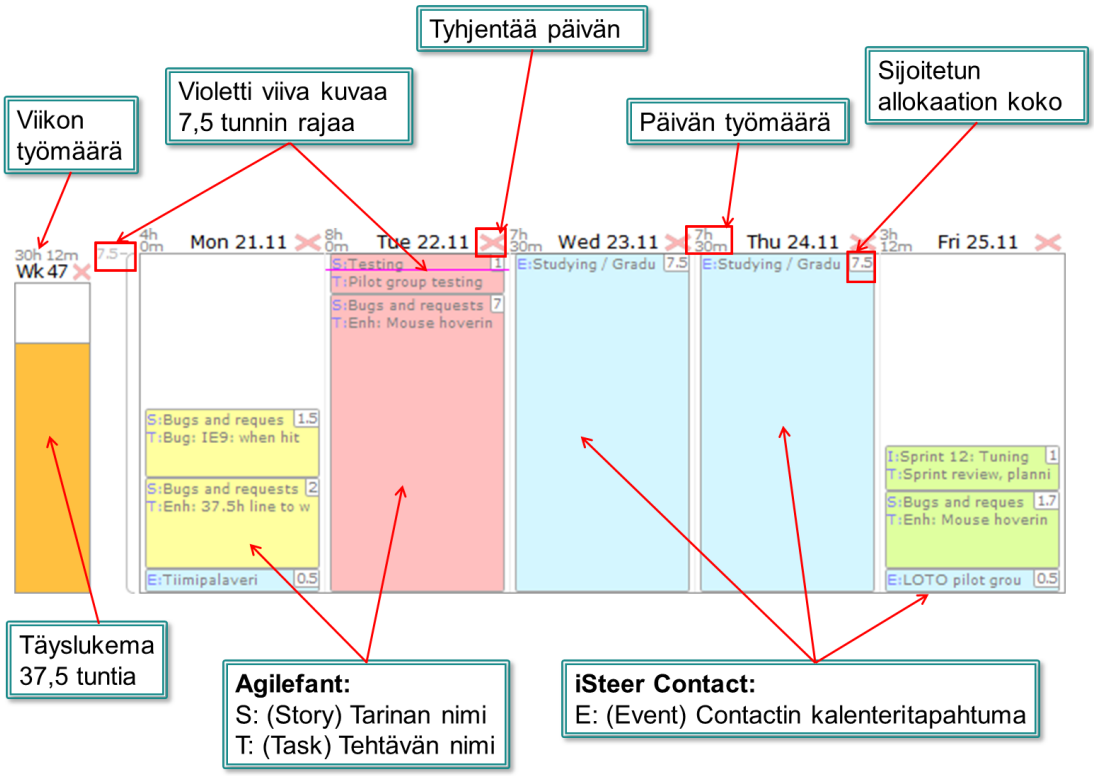
Käyttöliittymän malliluokat palautetaan jQueryn puolelle JSON-muodossa käyttäen Springiin integroitua Jackson JSON -suoritinta, joka tekee muunnoksesta yksinkertaista. Riittää, kun annotoi käsittelijässä metodin *@ResponseBody*-annotaatiolla. [9.]

Puunäkymään haetaan käyttäjän kaikki tehtävät sekä näiden kaikki isännät, mutta jos järjestelmässä on tarina ilman tehtävää, niin tätä tarinaa ei haeta. Tämä johtuu siitä, että Agilefantissa tarinoilla ei ole omaa arvioitua työaikaa, mutta tehtävillä on. Tästä huolimatta olisi voinut tehdä tarinat sijoitettaviksi LoTo:ssa, mutta tämä ei kuulunut tuolloin tavoitteisiin. Agilefantissa tehtävillä on oltava myös yli nolla minuuttia jäljellä arvioitua työaikaa tai niitä ei ladata LoTo:n puunäkymään. Tähän poikkeuksena voi

painaa kuvion 12 ylälaidassa näkyvää *Show No EL* -painiketta, jolloin näkyviin latautuvat myös ne tehtävät, joilla jäljellä oleva työkuorma on nolla. Lyhenne EL tarkoittaa *Effort left*. Jos työkuorma on Agilefantissa merkitty nollaksi, niin pääsääntöisesti työtehtävä on myös merkitty valmiiksi.

6.4 Työkuormakalenterin viikkonäkymä

Kuviossa 14 on työkuormakalenterin yksi viikko esitetty tarkentavien kommenttien kanssa.



Kuvio 14. Työkuormakalenterin viikko.

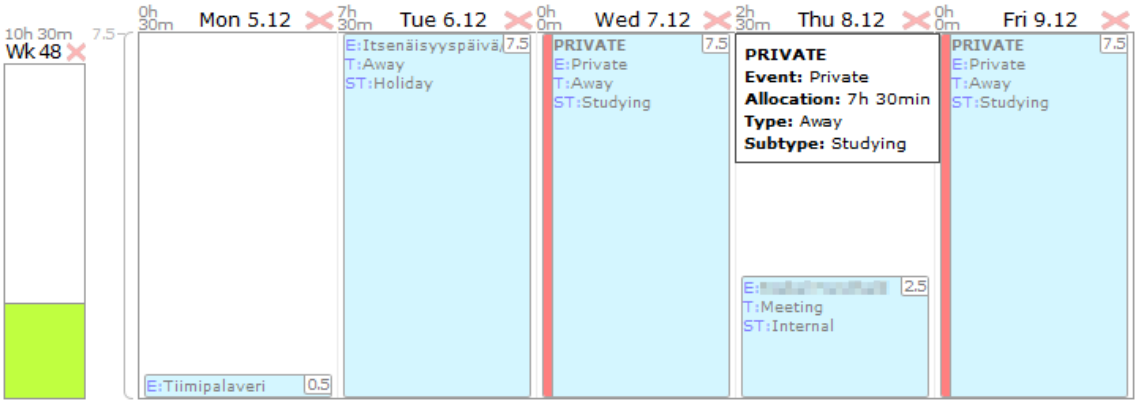
Kunkin merkinnän korkeus riippuu siihen sijoitetun ajan määrästä, joka näkyy näiden merkintöjen oikeassa ylänurkassa. Päivän yhteisestä työmäärästä riippuu Agilefantin tehtävien väri kalenterissa, joka kulkee työkuorman suuruudesta riippuen vihreästä punaiseen. Siniset merkinnät ovat IPSS iSteer Contactista ladattuja tapahtumia.

Koska merkinnän korkeus on riippuvainen sijoitetun työmäärän suuruudesta, tulee huomioida, että merkinnät eivät voi kasvaa yli päiväalueen rajojen. Tämä otetaan

huomioon mitoittamalla merkintöjen korkeudet päivän kokonaistyömäärän suhteen. Merkintöjen yhteisen työmäärän tullessa seitsemään ja puoleen tuntiin on päivä koko korkeudeltaan merkitty. Merkintöjen korkeus mitoitetaan uudestaan, kun päivän kohdalla sijoitukset ylittävät yhteensä seitsemän ja puoli tuntia tai ylittämisen jälkeen alittavat sen.

Päivälle voi sijoittaa rajattomasti lisää työmäärää, eikä tietenkään ole järkeä käyttäjän merkitä liikaa, mutta keinotekoisesti tätä ei rajoiteta. Tärkeänä tässä tulee kuitenkin esille, että jotenkin on ilmaistava päivä merkinnöissä, kun on sijoittanut yli tavallisen työmäärän ja tätä varten seitsemän ja puolen tunnin kohdalle ilmestyy silloin violetti viiva. Tämä on havaittavissa kuviossa 14 tiistaipäivän kohdalla, jossa on sijoitettu yli seitsemän ja puoli tuntia. Jos päivälle sijoittaa viisitoista tuntia yhteensä, niin violetti viiva sijaitsee keskellä päivää ja merkinnät on lyhennetty sopimaan päivän sijoituskehyksen alueelle.

Seuraavassa kuviossa on esitetty yksityistapahtumat.

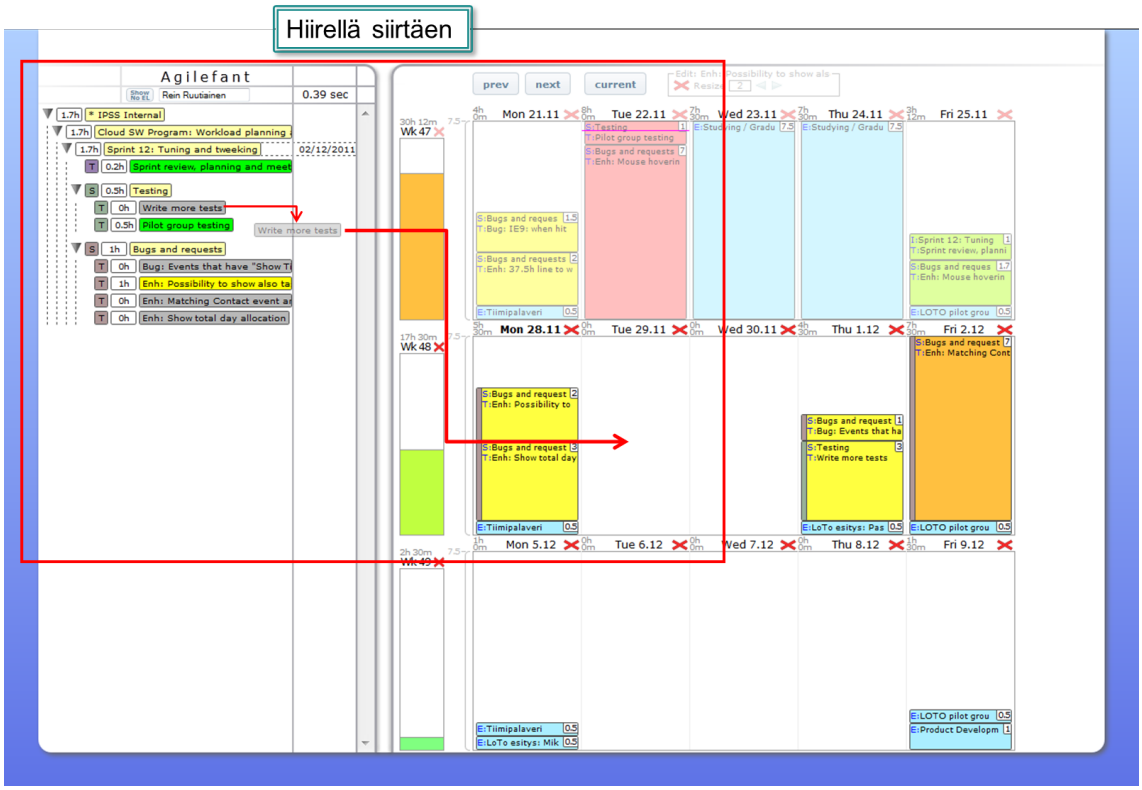


Kuvio 15. IPSS iSteer Contactactin kalenterin tiedoista on ladattu kaksi yksityis-tapahtumaa.

Päivän kokonaistyömäärä muodostuu aina sekä tehtävistä ja Contactin tapahtumista, paitsi että Contactissa yksityisiksi merkityt tapahtumat eivät vie työaikaa LoTo:ssa, kuten kuviossa 15 voi havaita Contactin yksityistapahtumista, joita merkitään punaisella reunalla.

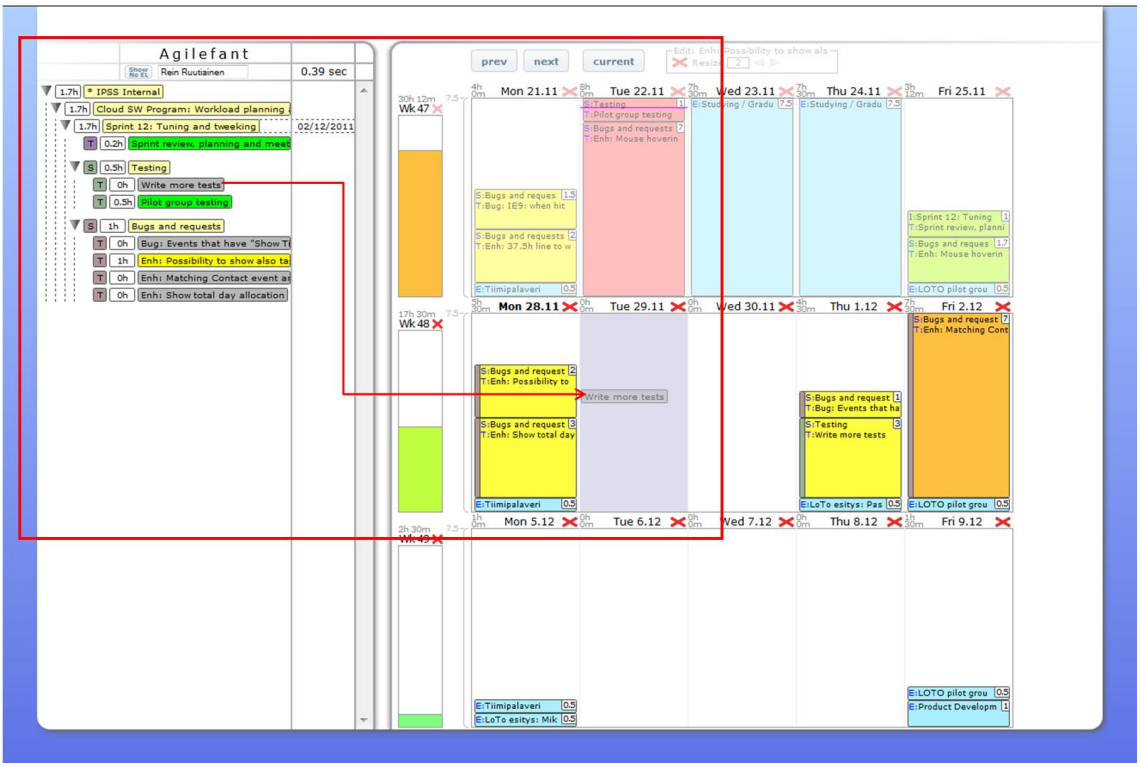
6.5 Tehtävän sijoittaminen puunäkymästä

Kuvioissa 16 ja 17 näytetään työtehtävän sijoittaminen työkuormakalenteriin.



Kuvio 16. Agilefantin tehtävän sijoittaminen työkuormakalenteriin.

Kalenteriin Agilefantin tehtävät sijoitetaan siirtämällä ne hiirellä puunäkymästä hiirellä vetäen halutun päivän kohdalle, kuten kuvioissa 16 ja 17 esitetään.

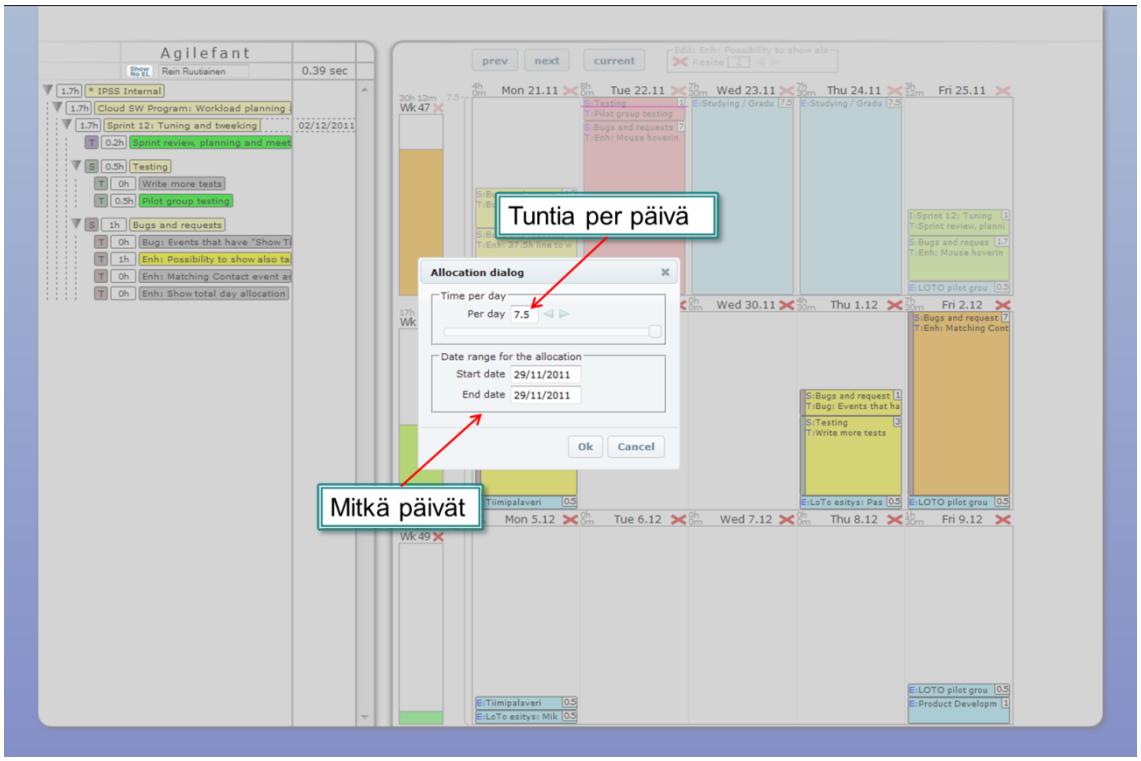


Kuvio 17. Agilefantin tehtävän sijoittaminen valittuun päivään.

Tehtävää hiirellä vetäessä korostuu hiiren kohdalla olevan päivän tausta tummemmalla värillä. Samoin myös koko viikon työkuormapylvään päälle vetäessä tehtävää tämän pylvään kehysalue korostuu.

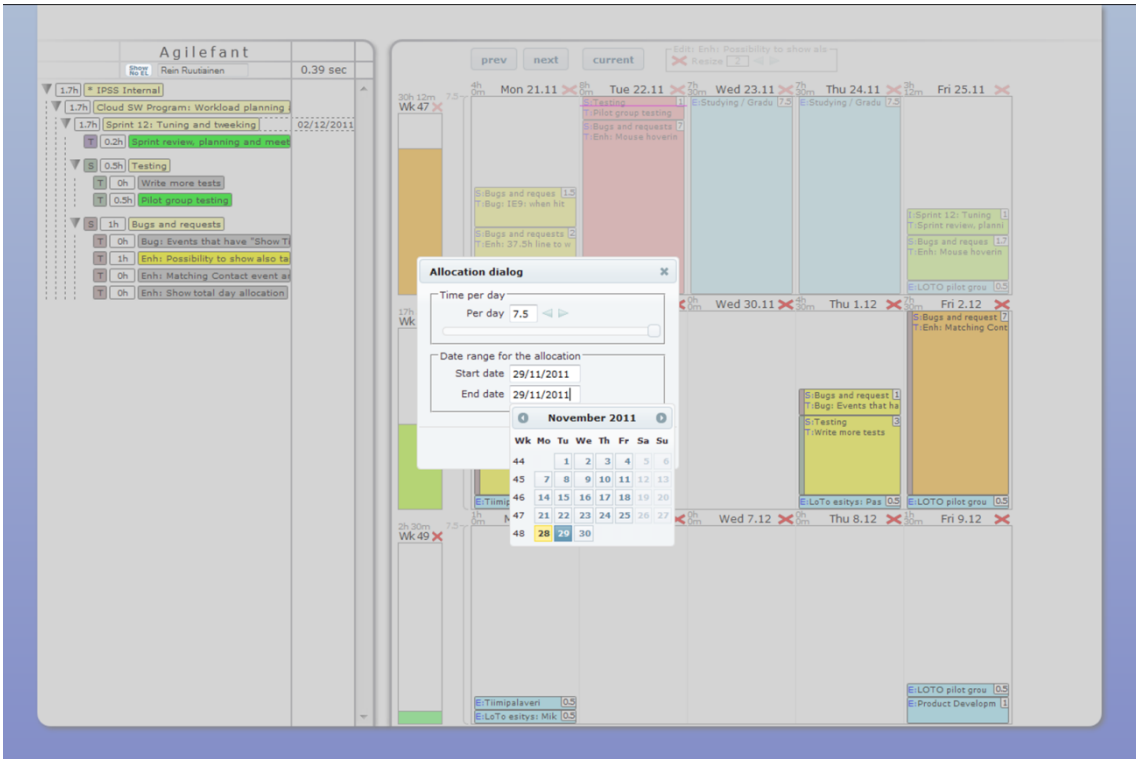
6.6 Sijoitusdialogi

Kuviossa 18 ja 19 käydään läpi sijoitusdialogin toimintoja.



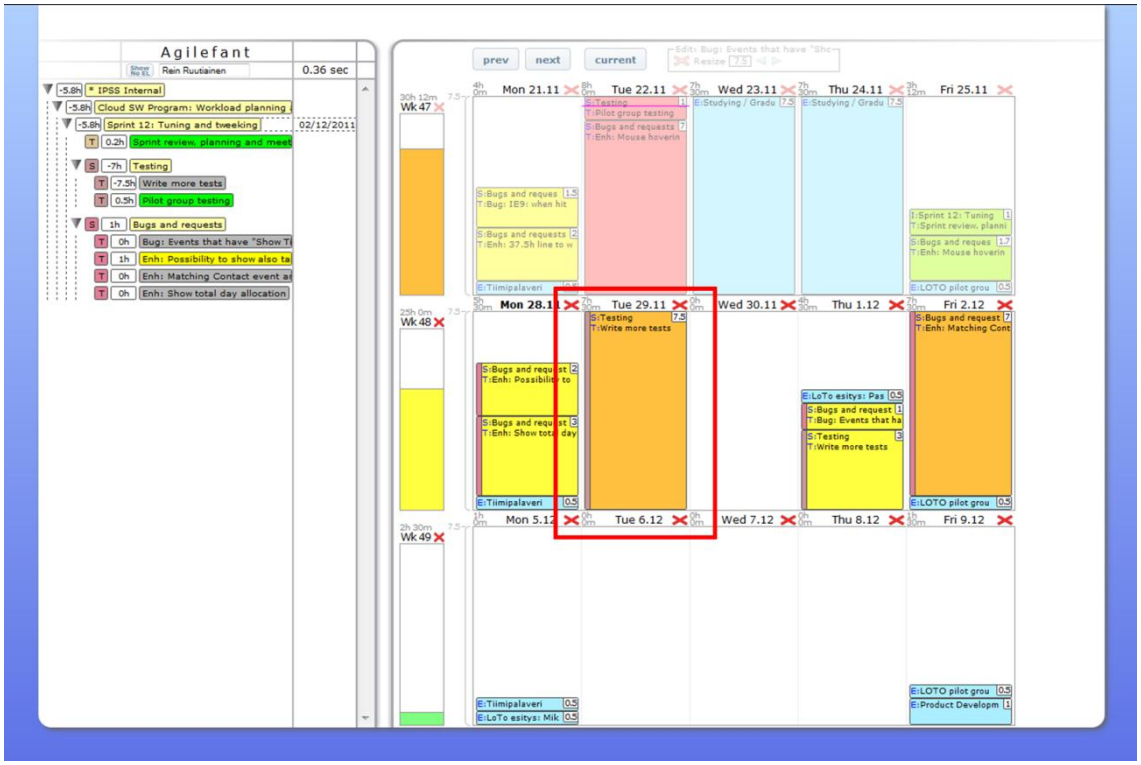
Kuvio 18. Agilefantin tehtävän sijoitusikkuna.

Tehtävän siirtämisen jälkeen tulee näkyviin sijoitusikkuna, jossa määritellään sijoituksen koko joko kirjoittamalla, liukusäädintä vetämällä tai nuolia painamalla. Liukusäädin sekä nuolet muuttavat aikaa puoli tuntia kerralla suuntaansa. Liukusäätimen maksimiarvo on seitsemän ja puoli tuntia tai *effort left* -arvon ylittäessä seitsemän ja puoli tuntia on maksimiarvo silloin sama kuin *effort left*. Sijoitetun arvon on oltava yli nolla tai muuten sijoitusta ei hyväksytä ja samaiseen dialogiin palautetaan virheteksti epäonnistumisesta. Sijoituksen suuruudelle ei ole kuitenkaan mitään rajoitusta eli voi kirjoittaa enemmän työaikaa tehtävään kuin jäljellä olevaa työkuormaa Agilefantissa on merkittynä. Puunäkymässä tämä ylisijoittaminen näkyy tehtävän *effort left* -tiedon kohdalla negatiivisena lukuna.



Kuvio 19. Agilefantin tehtävän päivien valinta sijoitusikkunasta.

Tehtävän voi sijoittaa kerralla niin monelle päivälle kuin haluaa ja määritetty tehtävän työaika siirtyy jokaiselle valitulle päivälle. Siirtäessä tehtävä hiirellä viikon työkuorma-pylvään päälle saadaan dialogiin valmiiksi määriteltyä kyseisen viikon kaikki arkipäivät sijoituskohteiksi. Minikalenterissa on deaktivoitu viikonloput eikä näille päiville voi sijoittaa. Viikonloppuja ei ole myöskään itse LoTo:n työkuormakalenterissa näkyvillä tai käytettävissä.



Kuvio 20. Näkymä uuden Agilefantin tehtäväsijoituksen jälkeen.

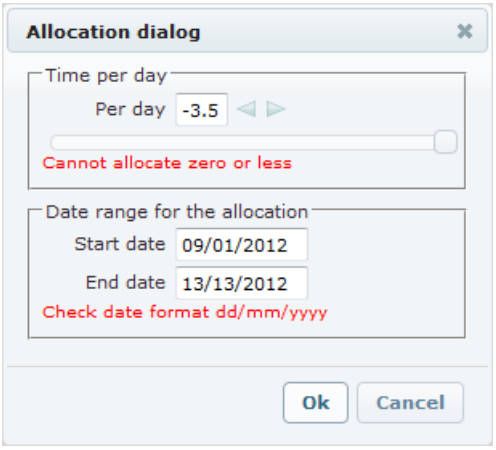
Kun sijoitus on hyväksytty, lähetetään palvelimelle AJAX-kutsulla sijoituksesta tiedot, kuten sijoituksen suuruus minuutteina ja aloitus- sekä lopetuspäivämäärät JavaScriptin Date muodossa. Sijoitus ilmaistaan desimaalilukuna tunneissa, mutta JavaScriptissa se muunnetaan ennen lähettämistä minuuteiksi ja pyöristetään kokonaisluvuksi.

Nämä AJAXilla lähetetyt parametrit yhdistyvät Springissä vaivattomasti suoraan metodin parametreihin, kun näissä käyttää *@RequestParam* -annotaatiota. Tällä tavoin ei tarvitse poimia tietoja *HttpRequestServletistä*.

Javan puolella annetaan minuuteille Integer- ja päivämäärille Date-tyyppi ja vahvistetaan, että minuutit ovat yli nollan. Päivämäärät vahvistetaan myös tarkistamalla näiden oikea muoto *dd/mm/yyyy,* sekä ettei päiviä kuukaudessa tai kuukausia vuodessa ole liikaa. Jos yhdessäkin annetussa parametrissa on virhe, niin näkymään palautetaan virheilmoitus, joka näytetään dialogissa oikeitten rivien kohdalla, eikä sijoitusta hyväksytä ennen kuin asia on korjattu. Päivämääräriville on mahdollista kirjoittaa vain numeroita ja vinoviivoja, sillä muut merkit on karsittu pois.

6.7 Sijoitusdialogin virheiden käsittely

Kuviossa 21 on esimerkki dialogin virheilmoituksista.



Kuvio 21. Dialogin virheilmoituksia.

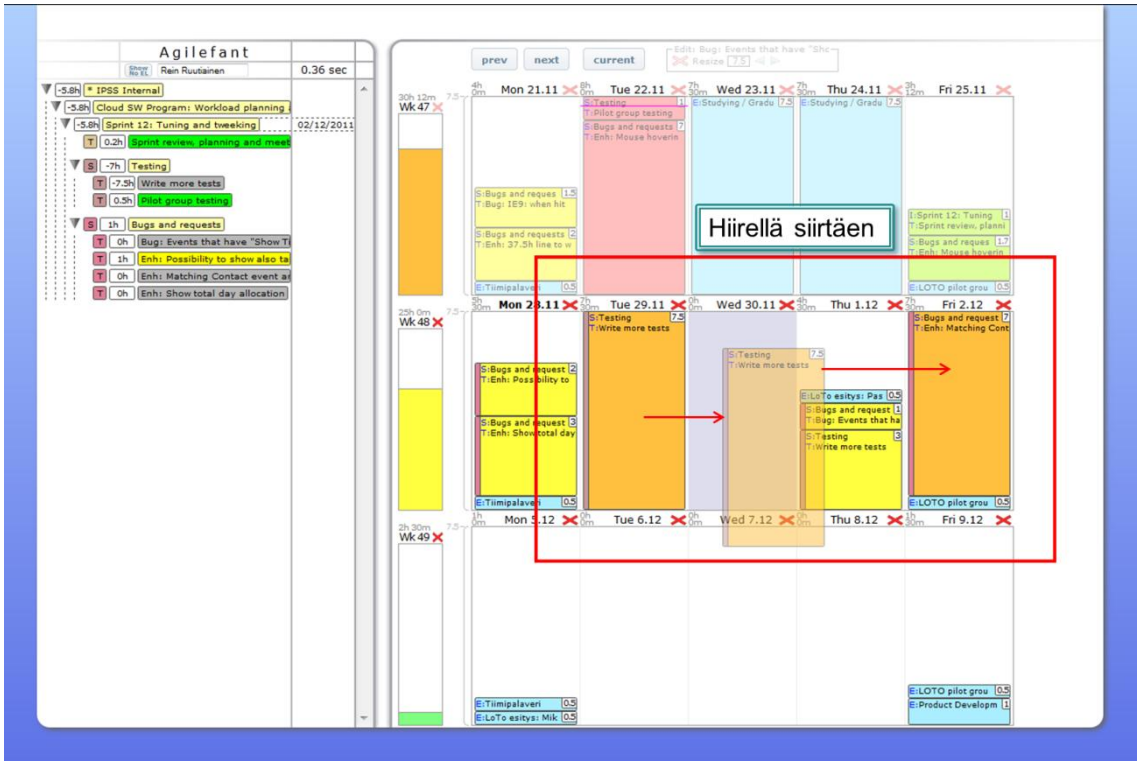
Aloitus- ja loppupäivämäärät merkitsevät, mille päiville tehtävän sijoitukset menevät. Tämä on toteutettu teknisesti niin, että LoTo:n kalenterissa jokaisessa päivän sijoitus alueen HTML-elementissä on jQuery data -tekniikalla tallennettu päivämäärä unixtime-muodossa, jossa aika on keskiyö. Unixtime kertoo ajan millisekunteina vuoden 1970 alusta laskettuna. Tämän perusteella päivät erotetaan toisistaan ja käyttäjän tallennetut LoTo suunnitelmat löytävät oikeilta päiviltä eli sama unixtime päivämäärä löytyy myös sijoitusten tallennuksista LoTo:n tietokannasta **allocations**-taulusta attribuutista Date.

Contactin tapahtumat tulee LoTo:n tietokantaan ulkoisesta lähteestä kopioituna. Päivämäärä on datetime-tyyppiä, eli siinä on mukana kellonaika. Tämä tarkoittaa, että aina tapahtumaa ladatessa tulee muokata päivämäärän kellonaika keskiyöhön, ennen kuin se muutetaan unixtime-muotoon.

Hyväksytyn vahvistuksen jälkeen aloitetaan uusien tehtävä sijoitusten luominen käyttäjäpuolella. Sijoitukset menevät valituille päiville ja tämän jälkeen mitoitetaan tarpeen mukaan kaikkien merkintöjen koot, kuten kuvion 14 jälkeen on kerrottu.

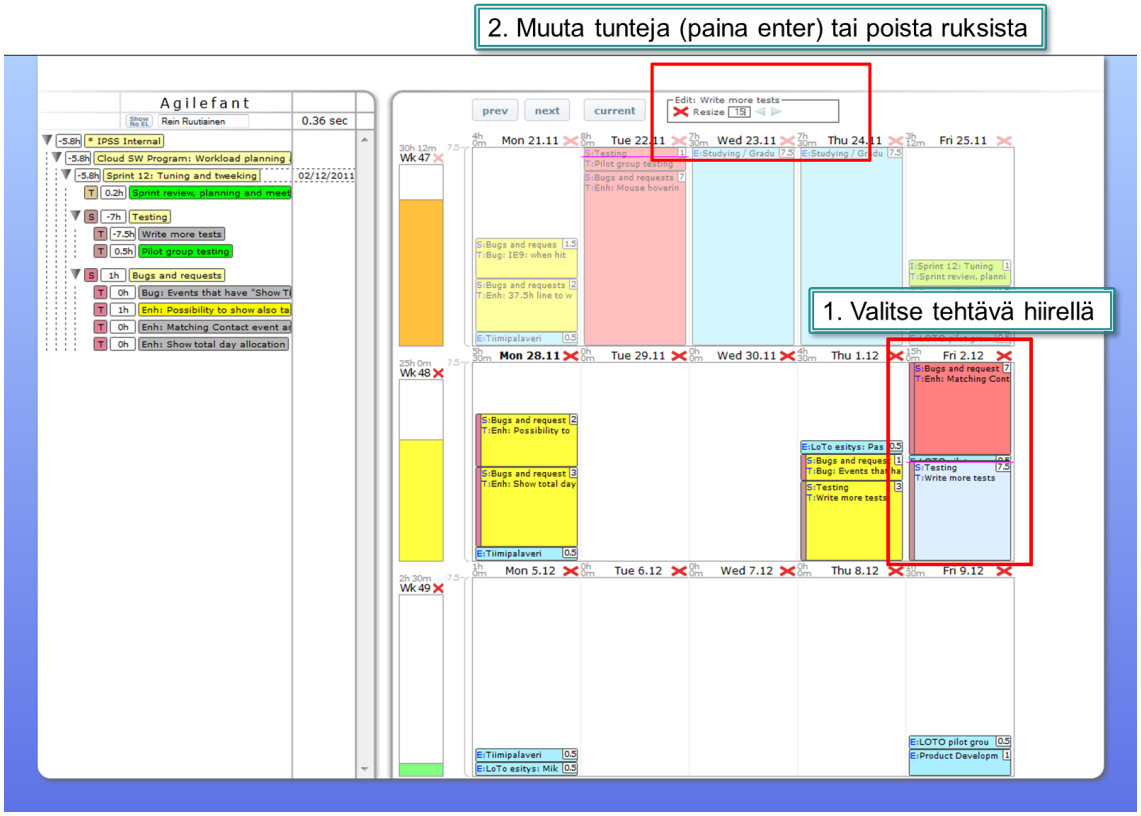
6.8 Tehtävän uudelleen siirto ja muokkaus

Tässä aliluvussa käydään läpi tehtävän siirto päivältä toiselle sekä sijoitetun tehtävän muokkaus.



Kuvio 22. Sijoitetun tehtävän siirtäminen toiselle päivälle.

Vain Agilefantin tehtävien sijoituksia voi sijoittaa uudelleen päivältä toiselle. Tämä tapahtuu siirtämällä hiirellä tehtävää halutun päivän kohdalle.

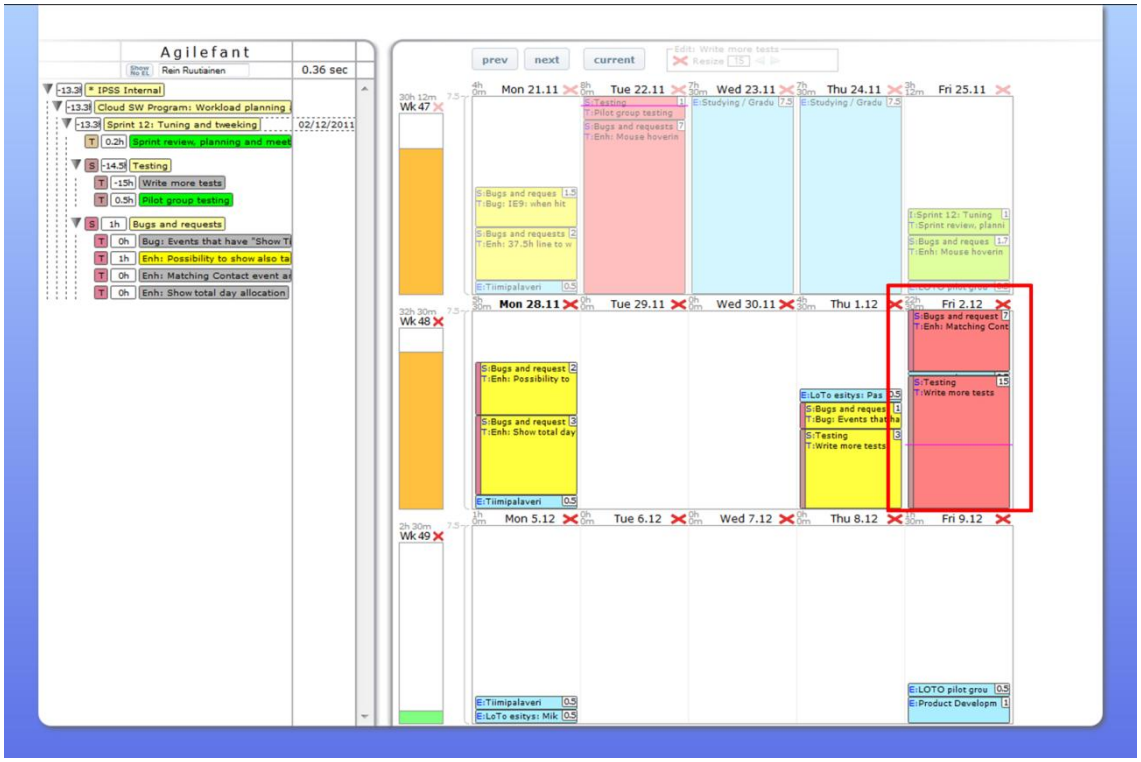


Kuvio 23. Agilefantin tehtävän valitseminen.

Tehtäviä voi valita painamalla hiiren vasenta painiketta tehtävän kohdalla, jolloin tehtävä muuttuu vaalean sinisiksi. Samalla kuvion 23 ylälaidassa osoitetussa kohdassa aktivoituu ruutu, josta valittua tehtävämerkintää voi muokata. Sijoituksen kokoa on mahdollista muuttaa kirjoittamalla tähän suoraan uusi aika. Myös viereisiä nuoli-painikkeita käyttämällä voi muuttaa haluamaansa suuntaan sijoitusta puoli tuntia kerrallaan. Ajan muutos vahvistetaan enter-näppäintä painamalla.

Vain yksi tehtävä voi olla valittuna kerrallaan. Jos yksi tehtävä on jo valittuna ja valitsee toisen, niin edellisen valintatila poistuu. Painamalla jo valittua tehtävää poistuu tästä valintatila. Punaisesta rastista tehtävämerkinnän voi poistaa.

Contactin tapahtumia ei voi ollenkaan muokata, poistaa tai siirtää. Ne pysyvät sellaisina kuin ne on Contactin puolella määritelty.



Kuvio 24. Agilefantin tehtävä työmäärä muokkauksen jälkeen.

Kuviossa 24 huomataan tehtävämerkintään tapahtuneen muutoksen muokkauksen jälkeen, kun tehtävän tunteja on lisätty. Kaikki tehtävät päivän kohdalla mitoittuivat tämän mukaisesti suhteellisesti oikeisiin kokoihinsa. Seitsemää ja puolta tuntia osoittava violetti viiva liikkui tämän mukaisesti myös oikealle paikalleen.

7 Yhteenveto

LoTo-projektissa kehitettiin pilottiversio työkalusta, jolla voi suunnitella ja hallita omaa työkuormaansa. Projektissa saavutettiin asetetut vaatimukset ensimmäisen version toteutukselle. Työkalu on nyt käytettävissä yrityksen sisällä ja käyttökokemuksia kuunnellaan potentiaalista jatkokehitystä ajatellen. Tämän hetkinen näkemys seuraavasta versiosta olisi muitten Agilefantin toimintojen integroiminen, kuten tuntikirjauksien tekeminen ja seuraaminen suoraan LoTo:sta.

Omat mahdollisuuteni vaikuttaa projektissa tehtyihin ratkaisuihin olivat hyvät. Projektin aikana sai vaikuttaa hyvinkin paljon käyttöliittymän suunnitteluun sekä teknisiin ratkaisuihin ja haastetta oli riittävästi, sillä toteutettavista asioista ei suuremmin ollut aikaisempaa kokemusta. Projekti kehitti yleisesti ymmärrystä verkko-ohjelmoinnista, eri suunnittelumalleista, ORM-tekniikasta ja MVC-arkkitehtuurista. Eniten projekti sisälsi käyttöliittymäpuolen toiminnallisuuksien suunnittelua ja toteutusta.

LoTo toteutettiin verkko-ohjelmaksi, jossa kaikki käyttöliittymäominaisuudet ohjelmoin jQueryllä. Käyttöliittymätoiminnot lähettävät tarvittaessa AJAX-kutsuja palvelinpuolelle, kun on esimerkiksi tarve tallentaa tai ladata tietoja tietokannasta. Palvelimella tiedot vastaanotetaan Java Spring MVC -moduulia käyttäen ja tietokantatoiminnot suoritetaan Hibernatella.

Lähteet

1 Lean Software Enterprise - Cloud Software Program. 2012. Verkkodokumentti. TIVIT Oy. <http://www.cloudsoftwareprogram.org/lean-sw-enterprise>. Luettu 7.2.2012.

2 TIVIT Oy. 2012. Verkkodokumentti. TIVIT Oy. <http://www.tivit.fi/fi/>. Luettu 7.2.2012.

3 Agilefant - Home. 2011. Verkkodokumentti. Aalto University, SoberIT. <www.agilefant.org/>. Luettu 7.2.2012.

4 SpringSource.org. 2012. Verkkodokumentti. VMware, SpringSource. <http://www.springsource.org/>. Luettu 7.2.2012.

5 jQuery Projects. 2012. Verkkodokumentti. jQuery Project. <http://jquery.org/>. Luettu 7.2.2012.

6 Hibernate - Jboss Community. 2012. Verkkodokumentti. Red Hat, JBoss Community Team. <http://www.hibernate.org/>. Luettu 7.2.2012.

7 Spring Framework Reference Documentation 3.0 - Introduction to the Spring IoC container and beans. 2010. Verkkodokumentti. SpringSource. <http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/beans.html>. Luettu 7.2.2012.

8 Donald, Keith. 2009. SpringSource Blog - MVC Simplifications in Spring 3.0. Verkkodokumentti. <http://blog.springsource.org/2009/12/21/mvc-simplifications-in-spring-3-0/>. 21.12.2009. Luettu 7.2.2012.

9 Donald, Keith. 2010. SpringSource Blog - Ajax Simplifications in Spring 3.0. Verkkodokumentti. <http://blog.springsource.org/2010/01/25/ajax-simplifications-in-spring-3-0/>. 25.1.2010. Luettu 7.2.2012.

10 jQuery UI - Home. 2012. Verkkodokumentti. jQuery Project. <http://jqueryui.com/>. Luettu 7.2.2012.

11 Piirainen, Tero. 2012. jQuery Tools – The missing UI library for the Web. Verkkodokumentti. <http://jquerytools.org/>. Luettu 12.3.2012.

12 Core J2EE Patterns - Transfer Object. 2010. Verkkodokumentti. Oracle Corporation. <http://java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html>. Luettu 7.2.2012.

13 Business Object. 2003. Verkkodokumentti. CoreJ2EEPatterns.com. <http://www.corej2eepatterns.com/Patterns2ndEd/BusinessObject.htm>. Luettu 7.2.2012.

14 Core J2EE Patterns - Data Access Object. 2010. Verkkodokumentti. Oracle Corporation.

<http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html>. Luettu 7.2.2012.

15 Bernard, Emmanuel. 2010. Hibernate Annotations Reference Guide - 2.2. Mapping with JPA (Java Persistence Annotations). Verkkodokumentti. <http://docs.jboss.org/hibernate/annotations/3.5/reference/en/html/entity.html#entity-mapping>. 15.9.2010. Luettu 7.2.2012.

16 Abram, Shaun. 2009. JavaBeans vs Spring beans vs POJOs. Verkkodokumentti. <http://www.shaunabram.com/beans-vs-pojos/>. 2.10.2009. Luettu 7.2.2012.

17 Poimala, Sami, Heikniemi, Jouni & Blåfield, Henrik. 2009. Ketterät käytännöt - Etusivu. Verkkodokumentti. <http://www.ketteratkaytannot.fi/fi-FI/>. Luettu 7.2.2012.

18 Poimala, Sami, Heikniemi, Jouni & Blåfield, Henrik. 2009. Ketterät käytännöt – Iteraatiot ja inkrementit. Verkkodokumentti. <http://www.ketteratkaytannot.fi/fi-FI/Ketteryys/IteraatiotJaInkrementit/>. Luettu 7.2.2012.

19 Poimala, Sami, Heikniemi, Jouni & Blåfield, Henrik. 2009. Ketterät käytännöt - Käyttäjätarinat. Verkkodokumentti. <http://www.ketteratkaytannot.fi/Kaytannot/Kayttajatarinat/>. Luettu 7.2.2012.

20 Seddighi, Ahmad Reza. 2009. Spring Persistence with Hibernate. Birmingham: Packt Publishing.